Oracle® Fusion Middleware

Developing with Oracle WebCenter Sites

12*c* (12.2.1.2.0)

E77465-03

Documentation for WebCenter Sites developers that describes how to design and build default and mobile websites with WebCenter Sites. This documentation also describes developing Visitor Services for capturing data across multiple external platforms to improve visitor segmenting and per-segment content targeting, developing A/B Testing for comparing individual pieces of content to determine effectiveness of each piece. This documentation also discusses conversions and events for better tracking and analysis of visitor activity.

ORACLE®

Oracle Fusion Middleware Developing with Oracle WebCenter Sites, 12*c* (12.2.1.2.0)

E77465-03

Contents

Part I Getting Started with Oracle WebCenter Sites

1 Introduction to Developing with WebCenter Sites

2 Overview of the Avisports Sample Site

3 The WebCenter Sites Development Process

Part II   Building Your Data Model

4   Understanding the Asset Types and Asset Models

5   Designing Basic Asset Types

6 Designing Flex Asset Types

7 Creating a Hierarchical Flex Family

8 Creating Flex Filters

9   Designing Attribute Editors

10   Configuring Bundled Attribute Editors

11   Working with the WebCenter Sites Database

12 Managing Data in Non-Asset Tables

Part III Developing a Website

13 Website Development with the MVC Framework and APIs

14 Developing a Server-Side Website

15 Developing a Client-Side Website

16 Website Development with Tag Technologies

17 About Sessions and Cookies

21 Coding Templates for In-Context and Presentation Editing

22 Template Element Examples for Basic Assets

23 Creating Collection Assets, Query Assets, and Page Assets

24 Best Practices for Creating Future Site Preview Assets and Templates

Part V Coding with Developer Tools

27 About Developer Tools

28 Installing and Configuring Developer Tools

29 Introducing Developer Tools Features in Eclipse

30 Developing JSPs with Developer Tools

31 Creating Templates for Mobile Websites Using Developer Tools

32 Synchronizing and Exchanging Data Using Developer Tools

33 Using Workspaces in Developer Tools

34 Using Developer Tools Command Line Interface (CLI)

35 Integrating Developer Tools Workspaces with Version Control Systems

36 Using Developer Tools to Manage and Exchange Resources

37 Using the Developer Tools Command Line Interface (CLI) to Create Reusable Modules

41 Using Advanced Page Caching Techniques

Part VII Migrating Your Work to Your Content Management System

42 Importing Assets of Any Type

43   Importing Flex Assets

44   Importing Flex Assets with the BulkLoader Utility

Part VIII   Security: Managing Content Management Users

45   Managing Users on the Management System

Part IX   Publishing Your Site

46   Publishing Your Content Management Site to Make it Available Online 46-1

Part X   Developing Personalized and Targeted Websites with Engage

47   Creating Visitor Data Assets

48 Understanding Recommendation Assets

49 Working with Memory-Centric Visitor Tracking

50 Coding Engage Pages

Part XI   Running A/B Testing

51   Developing for A/B Testing

Part XII   Customizing Blogs

52   Customizing Blog Components

Part XIII   Developing WebCenter Sites: Visitor Services

53   Developing WebCenter Sites: Visitor Services

Part XIV   Enabling Convergence between Eloqua and Oracle WebCenter Sites

54   Integrating Eloqua Cloud Marketing Service with Oracle WebCenter Sites

Part XV   Controlling the Site Capture Process

55   Coding the Crawler Configuration File

Part XVI Integrating with Third-Party Content Sources

56 Integrating Third-Party Content Sources Using Proxy Assets

Part XVII Developing Applications with the Web Experience Management (WEM) Framework

57 About the Web Experience Management (WEM) Framework

58 Understanding the WEM Framework and Services

59 Working with the Articles Sample Application

60 Developing Applications with WEM Framework

61 Developing Custom REST Resources with WEM Framework

62 Working with Single Sign-On for Production Sites

63 Using REST Resources with the WEM Framework

64 Introducing Customizable Single Sign-On Facility in WEM Framework

65 Buffering in WEM Framework

72 Customizing Global Properties, Toolbar, and Menu Bar in the Contributor Interface

73 Customizing Asset Forms for the Contributor Interface

74 Customizing Workflow

75 Working with RealTime Publishing Customization Hooks

76 Understanding Asset and Publish Events in WebCenter Sites

77 Customizing Content Audit Reports

78 Adding Customizations to WebCenter Sites

Part XIX Troubleshooting

79 Logging and Debugging Errors

Part XX Reference

80 Using Asset API: Tutorial

81   Using Public Site Search

Part XXI   Appendixes for Oracle WebCenter Sites Core

82   Introducing WebCenter Sites Tools and Utilities

83   Understanding White Space and Compression

84   Using WebCenter Sites URL Assemblers

Preface

This guide contains information about developing Oracle WebCenter Sites to support content contributors and administrators in creating, managing, and delivering highly interactive desktop and mobile websites.

Audience

This guide is written primarily for developers. It is assumed that developers have a clear knowledge of their company's business needs, and a basic understanding of their roles in the development of the online site and its back end. This guide is also useful to administrators, who collaborate with developers by setting up content management sites, site users, workflow processes, publishing methods, and Oracle WebCenter Sites client options.

Developers must know Java, JavaServer Pages (JSP), XML, and HTML. Administrators are not required to have programming experience, although a technical background is assumed.

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 that have purchased support 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

These additional documents may also be useful:

- *Oracle WebCenter Sites Release Notes*

- *Installing and Configuring Oracle WebCenter Sites*

- *Using Oracle WebCenter Sites*

- *Administering Oracle WebCenter Sites*

- *Property Files Reference for Oracle WebCenter Sites*

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 displays on the screen, or text that you enter.

Part I

Getting Started with Oracle WebCenter Sites

You'll be introduced to the tasks that you will typically perform in Oracle WebCenter Sites, tools and technologies that you will use, and development process you'll follow. You'll also become familiar with the sample site that comes packaged with WebCenter Sites.

Topics:

- Introduction to Developing with WebCenter Sites

- Overview of the Avisports Sample Site

- The WebCenter Sites Development Process

1

Introduction to Developing with WebCenter Sites

In WebCenter Sites, both templates and information are stored as assets. So, you begin with designing an asset model; creating asset types and assets. You also design site layout, page templates, and pagelets. For security and performance, you develop caching framework and security model. There is more to keep you engaged until you turn in the site to someone who'll administer it therefrom.

- About Developing with WebCenter Sites

- Typical Tasks for WebCenter Sites Developers

- WebCenter Sites Utilities

- WebCenter Sites Interfaces

- Use Case Scenarios for WebCenter Sites

1.1 About Developing with WebCenter Sites

Your role as a WebCenter Sites developer begins with building a core website, but it doesn't end here. You'll also tailor WebCenter Sites interfaces as the need arises. Does your company plan to leverage marketing-oriented components of WebCenter Sites? Extending these features to marketers so they can collect visitor profile information and design promotions for those visitors may be one of your key areas.

Your tasks as a developer can be grouped as follows:

- **Building a website**

  To create a website, developers build the website's infrastructure, administrators create content management site and site navigations, and content contributors add content to the website.

  This guide focuses on how developers can create a website's infrastructure using WebCenter Sites.

  In WebCenter Sites, both templates and information are stored as assets. To develop infrastructure of a website, first an asset model is designed which incorporates creating asset types and assets. Once asset types are ready, site layout, page templates, and pagelets are coded, and caching is implemented for better performance. For website access, users, ACLs, and roles are created, as well as the users are assigned to the relevant roles. Various types of content is imported as assets. The infrastructure is then mirror published to the management system where administrator and content contributors start designing the site. See The WebCenter Sites Development Process.

> **Note:**
>
> Depending upon an organization's setup, either developers or administrators create a content management site (framework to contain the content of an online site) and site navigations. Content contributors add pages and contents to the site navigations, and approve the content so administrator can publish it to the delivery system where the site goes online and starts functioning like a website.

For detailed information about building a website, see Getting Started with Oracle WebCenter Sites, Building Your Data Model, Developing a Website, Developing Mobile Websites, Coding with Developer Tools, Managing Caching, Migrating Your Work to Your Content Management System , and Security: Managing Content Management Users.

- **Enhancing the website**

  Websites can be enhanced depending on the nature of the business and customer profile. Using WebCenter Sites, online sites can be designed such that they gather visitor information and personalize promotional messages for each visitor, capture data about website visitors and their usage of pages. Site pages can be integrated with Facebook, Twitter, Google, and so on, as well as gadgets and applications can also be designed and integrated with WebCenter Sites.

  For detailed information, see Developing Personalized and Targeted Websites with Engage, Running A/B Testing, and Developing WebCenter Sites: Visitor Services.

- **Customizing WebCenter Sites**

  To make the development environment and experience efficient and the content contributor's job easier, you can customize the **Oracle WebCenter Sites: Admin** and **Oracle WebCenter Sites: Contributor** interfaces. You can alter properties, the dashboard, search views, asset forms, workflows, and so on for higher efficiency and productivity.

  For detailed information, see Customizing Oracle WebCenter Sites.

1.2 Typical Tasks for WebCenter Sites Developers

Some of the tasks you accomplish to build your core website are designing your site's data model, forms for users to enter information, sample assets, templates to display content assets, caching for performance.

See these topics for typical developer tasks:

- Data Models for Content Display

- Content Entry Forms for Content Management Sites

- Templates and Elements to Render Content on the Website

- WebCenter Sites Systems for Development, Management, Delivery, and Testing

- Approvals and Publishing

- Caching to Optimize Performance

1.2.1 Data Models for Content Display

WebCenter Sites developers build a data model for the content they need displayed on their website. WebCenter Sites supports the following data models:

- **Basic Asset Model**: This supports a flat data structure, so basic assets cannot inherit each other's properties (called attributes in this guide). Content is entered by WebCenter Sites users and is stored as objects called *assets* in the WebCenter Sites database. Each type of asset is contained in one primary storage table in the database, such that basic assets of one type can be associated with basic assets of another type.

- **Flex Asset Model**: This is a comprehensive data model in which each asset type uses several storage tables such that hierarchical data structures can be created, and child assets inherit attribute values from their parent assets. The flex asset model also supports flat data structures, within its own framework. Note that the flex asset model functions independently of the basic asset model; tables created within the two models do not intersect.

Whether you choose the flex asset model or the basic asset model depends on the complexity of the data you plan to serve to your visitors. The flex asset model has historically been used for creating large online catalogs of products. However, it can be used in less complex situations, and is especially desirable when the intent is to eventually convert flat data structures to hierarchical structures. The conversion process does not require you to re-create the data.

1.2.2 Content Entry Forms for Content Management Sites

WebCenter Sites developers use data models to create the content entry forms that contributors use to create content for the website. Each field in a content entry form maps to a corresponding column in a database table (or multiple tables). In addition, developers create the JSPs that render content entry forms in Web Mode and render published content on the website.

When content is ready for public delivery, it can be published to the website using either dynamic or static publishing. Formatted content is displayed on the website by JSPs. This table describes the difference between a dynamic WebCenter Sites page and typical HTML page.

Table 1-1 Static and Dynamic Pages

Static Page (HTML Page)	Dynamic Page (WebCenter Sites Page)
Single disk file, served by a web server.	Composed and created upon request.
One-to-one association between the HTML page and the page the visitor sees in the web browser.	The web page that the visitor sees can be composed of multiple components called *pagelets*, created from within WebCenter Sites.
No separation of presentation and content. As a result, it is difficult to modify presentation and content independently of each other.	Separation of presentation and content. As a result, presentation and content can be modified and maintained independently of each other.

1.2.3 Templates and Elements to Render Content on the Website

WebCenter Sites developers use APIs and JSP tags to code templates and elements used to render content on the website. The following programming components are used in the process of coding:

- Element Files

- APIs and JSP Tags

- Sessions and Cookies

1.2.3.1 Element Files

In very simplistic terms, the main function of WebCenter Sites is to separate format from content. By separating the two, WebCenter Sites enables you to reuse the same bits of formatting code for many pieces of content. For example, to change the format of articles, you rewrite the code in one place, rather than having to rewrite code for every article in your system.

Your formatting code is stored in files called *elements*. The code extracts the content from the database and formats the content. Because content is formatted only when a page is requested, you have the opportunity to design pages that will be constructed on-the-fly, according to the identity of the visitor requesting them.

Element files are stored in the `ElementCatalog` table in the WebCenter Sites database. The names of your pages are stored in the `SiteCatalog` table. That is, the `SiteCatalog` table stores the entries for all the legal page names for your website. Each row in the `SiteCatalog` table is a page entry. Each page entry points to an element in the `ElementCatalog` table. The element being pointed to by a page entry is called the root element of the page entry.

WebCenter Sites renders your content into an online page by executing `SiteCatalog` page entries. Here is how it works:

1. A visitor enters a URL to your website in a browser.

2. The web server that processes the HTTP request maps that URL to a WebCenter Sites URL. For example, a WebCenter Sites URL would look like this:

    ```
    http://www.FiscalNews.com/servlet/ContentServer?pagename=FiscalNews/Home
    ```

    The text after a WebCenter Sites URL is called the pagename. In this example, the pagename is `Fiscalnews/Home`.

3. WebCenter Sites looks up the pagename in the `SiteCatalog` table, determines its root element, locates that element in the `ElementCatalog` table, and then invokes that element.

    The element is executed. Elements that are called from within the root element are executed in turn.

4. The results (images, articles, and so on, including any HTML tags) are rendered into HTML code and returned to the visitor's browser.

The result is a page that is dynamically rendered on demand.

1.2.3.2 APIs and JSP Tags

WebCenter Sites includes several tag families that you use to code your elements. The tag families enable you to identify, extract, and then display assets on your website. WebCenter Sites also provides Java methods and utilities that you can use for designing your website, for developing your own content management applications, and for customizing the WebCenter Sites modules/products.

For information about coding pages that display assets that use the basic data model, see Coding Elements for Templates and CSElements. For information about WebCenter Sites tags, see the *Oracle Fusion Middleware Tag Reference for Oracle WebCenter Sites Reference*.

The WebCenter Sites operating system consists of several servlets that run on top of an application server. Each servlet is invoked when necessary to perform a discrete set of tasks. Each servlet has a corresponding Java API with Java methods and JSP tags that you use to invoke the functions.

This figure shows the main WebCenter Sites servlets:

Figure 1-1   Main WebCenter Sites Servlets
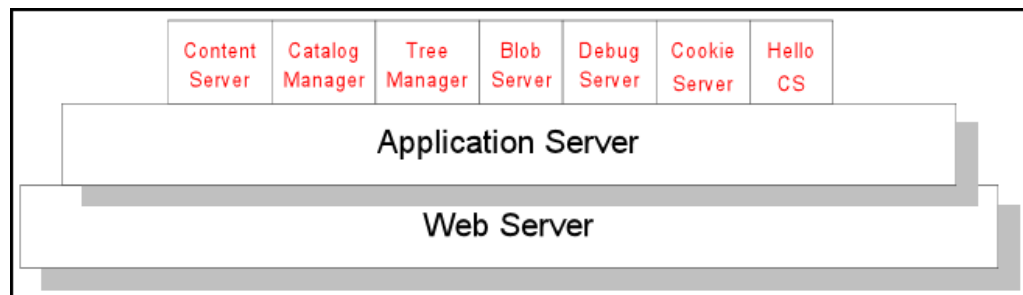
The main WebCenter Sites servlets are as follows:

- **ContentServer**: Generates and serves pages dynamically. This servlet provides disk caching, session management, event management, searching, and personalization services.

- **CatalogManager**: Provides most of the database management for the WebCenter Sites database, including revision tracking, security, resultset caching, and publishing services.

- **TreeManager**: Manages the tree tables, which store hierarchical information about other tables in the WebCenter Sites database.

- **BlobServer**: Locates and serves binary large objects (blobs). Blobs are not processed in any way. They are served as is, as they are stored.

- **DebugServer**: Provides tools that help you debug your XML code.

- **CookieServer**: Serves cookies for WebCenter Sites pages, whether those pages are delivered by the ContentServer servlet or by the Satellite Server application.

- **HelloCS**: Displays version information about the WebCenter Sites software installed on your system.

In general, you do not have to know which servlet performs which service or task. You simply invoke the appropriate Java method or XML or JSP tag and let the WebCenter Sites core application determine which servlet to call. The exception to this

rule is when you write code that references a servlet URL. That is, when you include a link to a blob or to another page on a WebCenter Sites page. Because the ContentServer servlet and the BlobServer servlet reside at different URLs, you must include the URL of the appropriate servlet in your `<A HREF>` tags.

For information about the coding links to blobs and pages, see Website Development with Tag Technologies and Coding Elements for Templates and CSElements.

1.2.3.3 Sessions and Cookies

WebCenter Sites automatically creates a session for a visitor when he or she visits your website for the first time. You can store information about that visitor in session variables by using the tags and methods in the WebCenter Sites core. Subsequent elements can then access those variables and respond conditionally to them.

Session variables, however, are volatile. They last only while the session lasts, that is, until one of the following events occurs:

- The visitor closes his or her browser.

- The session times out after a period of inactivity. You control session timeouts by setting the `cs.timeout` property (in the `wcs_properties.json` file) from the Property Management Tool in the Admin interface.

- The application server is restarted (except in a cluster).

- The session is disabled in some other way.

Cookies are used to store information in a more permanent manner. You can code your elements to write cookies that store information about your visitors to their browsers. Then, you can use the stored information to customize pages and display the appropriate version of a page to the appropriate visitor when he or she returns to your website.

See About Sessions and Cookies.

1.2.4 WebCenter Sites Systems for Development, Management, Delivery, and Testing

When you are working with WebCenter Sites for your content management needs, you and the others on your team work with up to four different systems:

- **Development System**: Where developers and designers plan and create the website. All of the WebCenter Sites products that you have purchased are installed on this system.

- **Management System**: Where content providers such as writers, editors, graphic artists, and marketers are assigned to content management sites to develop the content that is delivered to visitors of the website. Revision tracking and workflow features track changes to assets (content), monitoring them until they are approved to be published to the delivery system.

  Content management sites represent the real website. For example, you could create separate content management sites for separate sections of your website because the teams who provide content for each section work completely separately from each other and only members of that team should have access to that section (content management site). Or, you could create a content management site that represents an entire website, as does the avisports sample site. See Assembling Content Management Sites in *Oracle Fusion Middleware Administering Oracle WebCenter Sites*.

- **Delivery System**: Here the content you are making available or the products that you are selling are served to your visitors or customers.

  To deliver content dynamically, you should install all of the WebCenter Sites products that you purchased on this system. To deliver content statically, that is, to serve static HTML pages, your delivery system needs a web server only. That is, you don't have to install any of the WebCenter Sites products on your system.

- **Testing System**: Where you or your QA engineers test the performance of both the management system and the delivery system. Testing can be performed on either a dedicated system or on the development system itself.

WebCenter Sites developers spend the majority of their time working on the development system. When the asset types that you develop and the site that you have designed are ready, you migrate (publish) your work from the development system to the management system. As assets are created, modified, and approved by the content providers, they are published from the management system to the delivery system.

1.2.5 Approvals and Publishing

When you finish developing the website, you publish your work (templates, elements, asset types, the site navigation, and so on) from the development system to the management system. Publishing your work makes it available on the management system. Contributors can then use the asset types and your site design to create content for the website. When contributors are finished creating the site content, that content (along with the supporting asset types, templates, elements, site navigation, and so on) can be approved and published to the website.

When assets are ready to be published, someone first marks them as approved. Then, when the publishing process is ready to start, it invokes the approval system which compiles a list of all the approved assets and examines all the dependencies for those assets. Assets linked to an approved asset must also be approved before the asset can be published.

The WebCenter Sites publishing and approval systems track and verify all the asset dependencies to maintain the integrity of the content on your delivery system. The publishing and approval systems ensure that the assets that are ready for publishing are the only assets that get published.

When you publish content and elements, WebCenter Sites copies them from one system (for example, your management system) to another system (for example, the delivery system). WebCenter Sites delivers two publishing methods that are built from the WebCenter Sites publishing APIs. These publishing methods interact with the WebCenter Sites approval system, an underlying system that determines which assets have been approved.

The WebCenter Sites publishing methods are:

- **RealTime**: The dynamic publishing method. It is built with the WebCenter Sites RealTime API to copy approved assets from the WebCenter Sites database on one system to the WebCenter Sites database on another system.

- **Export to Disk**: The static publishing method. It renders your approved assets into static HTML files, using the template elements assigned to them. An administrator or automated process then copies those files to your delivery system using FTP or another file transfer method.

> **See Also:**
>
> - Tips for Configuring Publishing Destination Definitions in *Oracle Fusion Middleware Administering Oracle WebCenter Sites* for information about configuring publishing
>
> - Coding Elements for Templates and CSElements for information about coding elements so that they log dependencies appropriately and how WebCenter Sites calculates approval dependencies
>
> - Approving and Publishing Content in
>
>   *Oracle Fusion Middleware Using Oracle WebCenter Sites* and Approving Multiple Assets in *Oracle Fusion Middleware Administering Oracle WebCenter Sites* for information about how to approve assets

1.2.6 Caching to Optimize Performance

Developers implement various caching frameworks to optimize the performance. WebCenter Sites also supports the use of Satellite Server caching, which provides a second level of caching and can also be used as a remote cache for your web pages. By default, WebCenter Sites and Satellite Server use inCache as their page caching framework. The following topics describe caching:

- Page Caching

- Resultset Caching

- Asset Caching

- Satellite Server Caching

1.2.6.1 Page Caching

Page caching is implemented at the template level and is used to cache pages on the WebCenter Sites system. Page caching plays a significant role in system performance. A cached page can be served much faster than it can if it must first be generated.

WebCenter Sites alone (independently of Satellite Server) can separately cache each page or pagelet that is identified by a page entry in the `SiteCatalog` table. You can mark the expiration date of any pagelet in the cache by specifying a value for that page entry in that table.

Page caching is made especially effective by the addition of Satellite Server. Installing a Satellite Server application amounts to installing page caches on the servers that host Satellite Server, thereby extending the WebCenter Sites page cache.

> **See Also:**
>
> - Understanding Page Design and Caching for information about page caching
>
> - Configuring Your System for inCache Page Caching in *Oracle Fusion Middleware Administering Oracle WebCenter Sites* for information about inCache page caching
>
> - Satellite Server Caching for information about Satellite Server

1.2.6.2 Resultset Caching

Resultset caching is another feature that can greatly enhance system performance. When the WebCenter Sites database is queried by any mechanism, the WebCenter Sites application can cache the resultset that it returns. It keeps track of every table in the database. Whenever a table is modified, it flushes all the resultsets that were cached for that table.

See Working with Resultset Caching and Queries.

1.2.6.3 Asset Caching

Asset caching is a memory-based system that is built on the inCache framework to optimize the performance of WebCenter Sites by taking up load that would otherwise affect the database. In WebCenter Sites, programmatic usage of assets consists of loading and rendering their attributes. Given that assets are loaded by templates, which are stored in the WebCenter Sites database, `AssetCache` is used only on WebCenter Sites nodes. Asset caching includes the `AssetCache` container component which functions by caching assets and interacting with existing inCache components.

See Using the inCache Framework in *Oracle Fusion Middleware Administering Oracle WebCenter Sites*.

1.2.6.4 Satellite Server Caching

Satellite Server is a caching application. It supplements WebCenter Sites caching functionality by providing additional page caches. The tandem use of the WebCenter Sites and Satellite Server caches results in automatic double-buffered caching.

By default, co-resident Satellite Server is installed on the same computer where WebCenter Sites is installed. You can further improve your system's performance by installing Satellite Server remotely so it can cache pages and pagelets closer to their intended audience. Remote Satellite Server hosts are fast, inexpensive caches of WebCenter Sites pages. They reduce the load on the WebCenter Sites host, dramatically increase the speed of page delivery to your site visitors, and provide a simple and inexpensive way to scale your WebCenter Sites system.

1.2.6.4.1 HTTP Requests

When the load balancer routes an HTTP request for a page to Satellite Server, Satellite Server either serves the page if the page is in its cache, or if the page is not cached, it forwards the HTTP request to WebCenter Sites. The basic chain of events is the following:

1. Satellite Server checks its cache.

2. What happens next depends on whether the page is in the Satellite Server cache (see Table 1-2 for details).

Table 1-2   Pages in or not in the Satellite Server Cache

Page in the Satellite Server Cache	Page Not in the Satellite Server Cache
Satellite Server serves the page to the visitor's browser. In this case, Satellite Server does not have to forward the request to the WebCenter Sites database, thus reducing the load on the database.	• Satellite Server forwards the request to WebCenter Sites. • WebCenter Sites returns the cached pages from its cache to Satellite Server. It renders the page which is not in its cache, caches a copy, and sends the page to Satellite Server. • Satellite Server then caches the page and serves it to the visitor's browser. When requested again, the page is served from the Satellite Server cache, which reduces the load on the WebCenter Sites database.

Each Satellite Server application is independent of every other Satellite Server application. An individual Satellite Server application has the following characteristics:

- It maintains its own cache.

- It cannot request pages or pagelets from another Satellite Server application. It can request pages or pagelets from only the WebCenter Sites core.

1.2.6.4.2 Satellite Server Servlets

Satellite Server is made up of several servlets: one that caches and serves pages, and two that manage the cache:

- **Satellite**: Caches pages at the pagelet level. The Satellite XML or JSP tags in your elements indicate which pagelets should be cached, and they control various Satellite Server settings.

- **Inventory**: Enables you to examine the Satellite Server cache so you can obtain the information you need to manually flush individual pages or pagelets from the cache when necessary.

- **FlushServer**: Handles all types of cache-flushing. FlushServer can either flush the entire cache, or can flush individual items from the cache.

For information about coding pages with the Satellite Server tags and page caching in general, see Understanding Page Design and Caching.

1.3 WebCenter Sites Utilities

Many GUI-based WebCenter Sites utilities are available for you to manage the WebCenter Sites database and various code. Decide which utility you need and install it on your system.

- Developer Tools, which integrates WebCenter Sites with the Eclipse Integrated Development Environment (IDE). The Developer Tools kit enables WebCenter Sites developers to work in a distributed environment using tools such as Eclipse and version control system (VCS) integration.

- Sites Explorer, for viewing and editing tables in the WebCenter Sites database.

- CatalogMover, for exporting and importing database tables.

- XMLPost, for incrementally importing data into the WebCenter Sites database.

- BulkLoader, for quickly importing large amounts of data into the WebCenter Sites database.

- Property Management Tool, accessible from the Admin interface, for viewing and organizing the `wcs_properties.json` file (system configuration files).

1.4 WebCenter Sites Interfaces

You'll use the Admin interface to accomplish several different tasks. However, it's a good idea to get familiar with Contributor and WEM interfaces, too.

- **Admin Interface**: The Admin interface allows developers and administrators to manage and configure WebCenter Sites.

Figure 1-2   Admin Interface
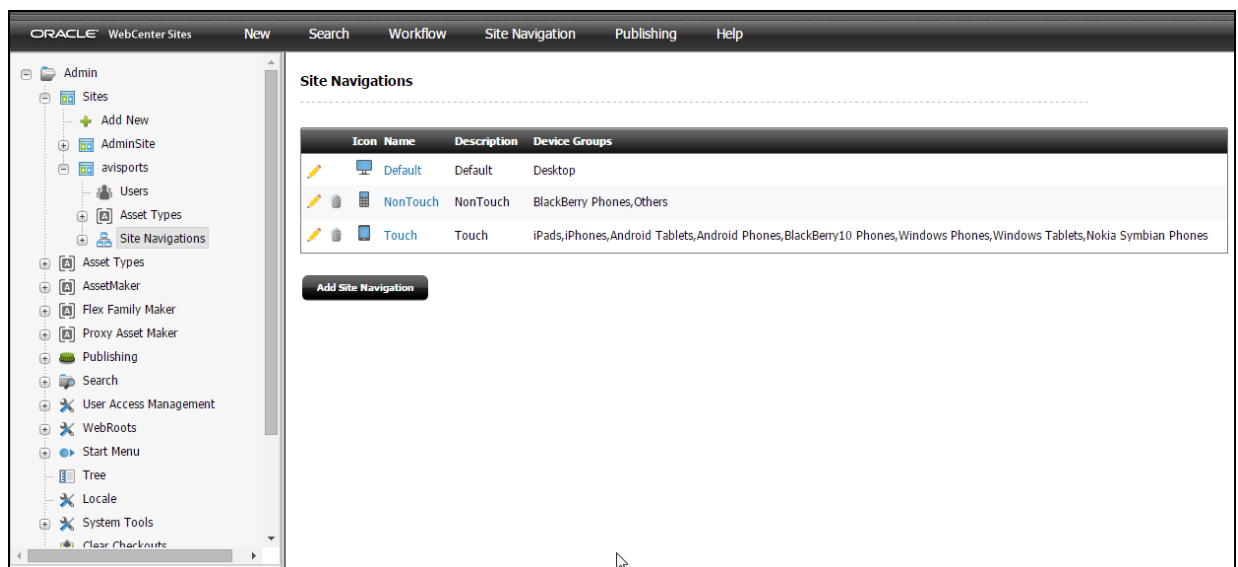
The tree panel on the left contains all the content management elements that developers and administrators have to work with. The workspace area on the right is where all the tasks and operations are performed.

The Admin interface supports code-based operations, and enables you to graphically complete the creation of basic asset types. For example, to create a basic asset type, you would:

1. Write an XML file (called *asset descriptor* files) to define the basic asset type.

2. Upload the file to WebCenter Sites.

3. Invoke the AssetMaker utility. One of the functions of the interface (AssetMaker) is to read the asset descriptor file and, from it, create a storage table for the asset type. Other functions in the interface allow you to configure the asset type (for example, name its authorized users).

The same interface is used by administrators to create content management sites, manage system users, control their permissions to content, establish workflow processes, and configure WebCenter Sites features (such as Mobility).

- **Contributor interface**: The Contributor interface is designed specifically for content providers and business users. It provides ease of use and quick access to most WebCenter Sites content management functions, such as previewing, creating, editing, deleting, and approving assets.

Figure 1-3   Contributor Interface
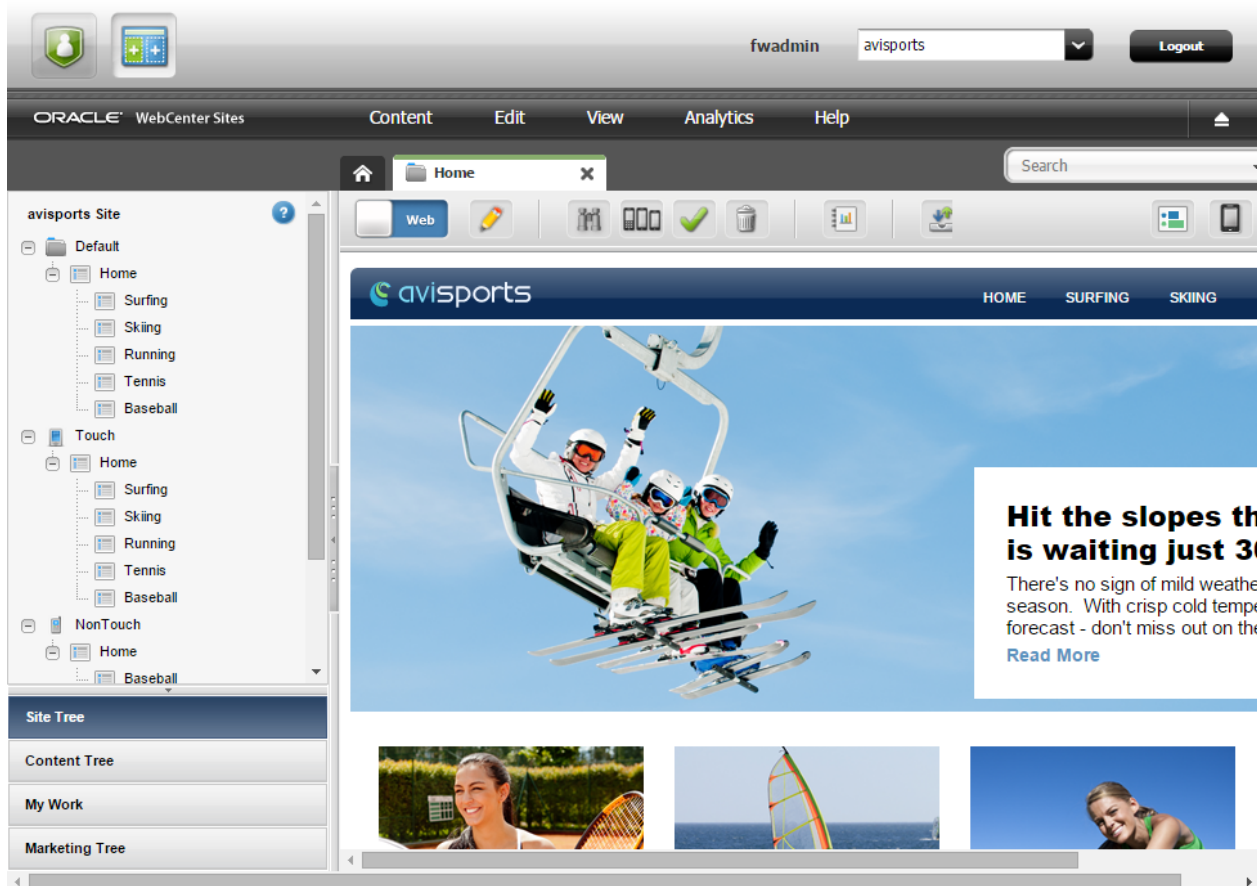
When you work with assets in the Contributor interface, you may see fields enabled with the following WYSIWYG editors:

- **CKEditor**: An open source WYSIWYG text editor from CKSource which requires no client-side installation. Developers can use CKEditor to create basic assets whose text-entry fields use CKEditor as the input mechanism for the field. Developers can also create attribute editors for flex attributes that use CKEditor as the input medium.

- **Clarkii Online Image Editor (Clarkii OIE)**: A popular third-party image editor from InDis Baltic. Developers can enable Clarkii OIE to allow users to edit images directly in the Form Mode eliminating the need for an external image editor.

- **WEM Admin Interface**: The WEM Admin interface is designed specifically for administrators to manage the assignment of applications and users to sites using roles.

Figure 1-4   WEM Admin Interface
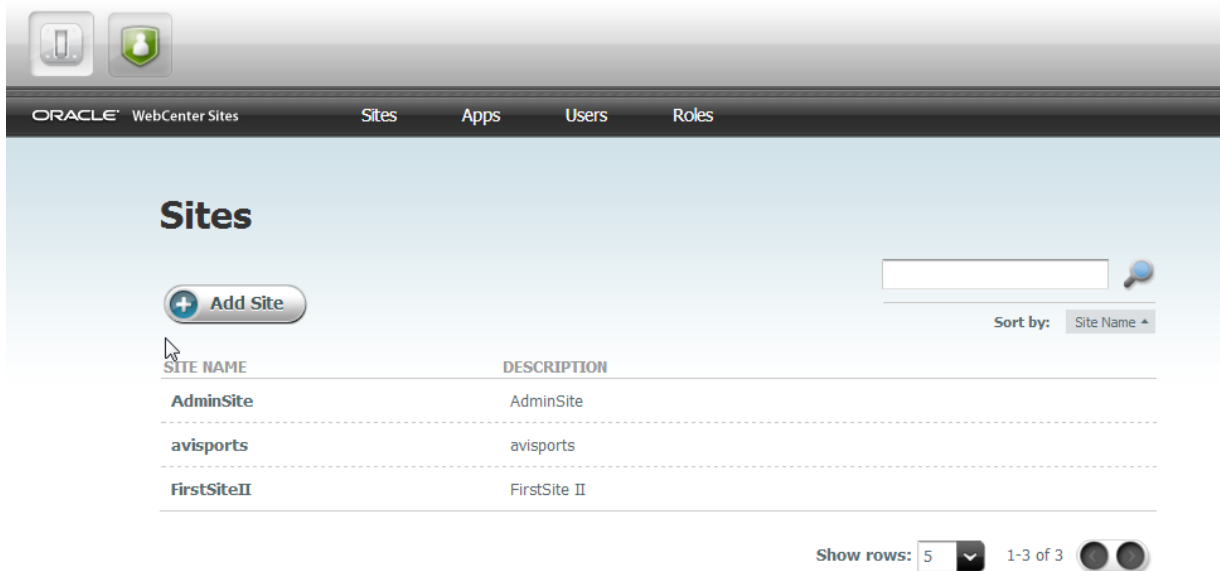
> **See Also:**
>
> Using the Web Experience Management Framework in *Oracle Fusion Middleware Administering Oracle WebCenter Sites* and Developing Applications with the Web Experience Management (WEM) Framework.

1.5 Use Case Scenarios for WebCenter Sites

WebCenter Sites provides capabilities for business-user content authoring, delivery of high-scale dynamic sites, content targeting and optimization, user-generated content, end-user personalization, marketing and lead generation, and mobile Web delivery. WebCenter Sites is used in a variety of industries to create informational and branding websites that run marketing campaigns and generate business leads.

These topics describe WebCenter Sites use cases:

- Developing Informational (Branding) Websites
- Creating Marketing-Oriented Websites
- Creating Mobile Websites

1.5.1 Developing Informational (Branding) Websites

WebCenter Sites provides easy-to-use and efficient features to develop branding websites for products and services. The starting point is creating the basic infrastructure using the WebCenter Sites core. Consider the following when designing a website with WebCenter Sites:

- **Content Type**: The first thing to determine is how the site content should be categorized and designed in WebCenter Sites. Which content type should be structured and which should be binary? Content that content contributors create

in WebCenter Sites through Form or Web mode is structured, but imported content (such as Microsoft Word files) is binary.

Architects determine the following about content types:

– Which content types should have variable attributes and which should have fixed attributes? A product model or service type of content may require variable attributes as companies improve their existing range of offerings from time to time.

– Which content should be flat and which should be hierarchical? For instance, images are usually flat or basic type. A product accessory model such as headphones for a MP3 player can be hierarchical.

– Based on the product or service, which content should inherit attributes from other content? A product model may need to inherit attributes from the parent product. Some content types may be standalone.

– Some content types require associated content. For example, you may need to associate an article about a product model with articles about similar product models or the parent product.

– How will contents be recovered if a disaster occurs?

These considerations determine the asset model and its implementation. Typically, websites require a combination of basic and flex assets. For information about how content types are determined and designed in WebCenter Sites, see Understanding the Asset Types and Asset Models.

- **Content Volume**: A website should be designed to handle any volume of content. It should be scalable.

- **Pages**: A page contains many pagelets that can be reused. When designing pages consider reusability benefits as well as caching strategy for better performance. See Coding Elements for Templates and CSElements and Understanding Page Design and Caching.

- **Page Caching**: For better performance, determine when pagelets will be used. Design page templates to use fewer uncached pagelets per page. See Managing Caching.

- **Templates**: Before designing templates, consider those scenarios when template components might be reused; pagelets can be reused on other pages. Some examples of pagelets are: Top, LeftNav and Footer as they are likely to be reused on many pages. See Developing a Website.

- **Content Mode**: WebCenter Sites lets you include Form mode and Web mode in your content management site. While Form mode is quick to use, Web mode enables infrequent users or users who perform a limited role to find, edit, and submit content directly from the rendered (Preview) version of an asset. Web mode also enables content contributors to compare two or more versions of a web page to determine which version is more effective. See Developing a Website.

- **Content Management (CM) Sites**: A CM site is the source of content for the online site and can represent either an entire online site or one of its sections. Consider customers' needs and determine how CM and online sites should be designed: whether a single CM site and its single online site, or a single CM site and multiple online site, or multiple CM sites and multiple online sites. See

Content Management Models in *Oracle Fusion Middleware Administering Oracle WebCenter Sites*.

- **Multilingual Requirements**: Find out what customers are looking for. Mostly customers want the ability to create content in a master language, wire it up to other related content (which may or may not be in the master language), then either publish it and translate it later. Or, they prefer translating the content and other related bits and pieces first, and then publishing the whole as a single package. Determine if customers want certain country-specific rules to apply to the rendered content. See Configuring Sites for Multilingual Support.

- **User permissions**: You use Access Control Lists (ACLs) to restrict user access to the WebCenter Sites database and the rendered pages served on your sites by WebCenter Sites. WebCenter Sites also provides user tags to log in and log out users, as well as to create an account or edit user profiles. Some of the common user permissions are: create, edit, delete, approve content, rights to access WEM, Admin, and Contributor interfaces. Consider what all types of permissions should be created for different roles. See Creating and Authorizing Users in *Oracle Fusion Middleware Administering Oracle WebCenter Sites* and Security: Managing Content Management Users.

- **Security**: Before developers begin designing the online site or contemplating changes to the user interface on the management system, you must determine and implement your security protocols. The decisions you make about security configuration affect the way that you code and implement your online site. See Setting Up External Security in *Oracle Fusion Middleware Administering Oracle WebCenter Sites*.

- **Customization**: Customer teams interact with the Contributor interface to edit and update websites. For their efficiency and convenience, you can customize the Contributor interface components such as the site tree, dashboard, asset forms, search views, and so on. See Customizing Oracle WebCenter Sites.

1.5.2 Creating Marketing-Oriented Websites

Oracle WebCenter Sites enables marketers and business users to easily create and manage contextually relevant website content aimed toward sales and customer loyalty. It provides components that let you develop personalized and targeted websites, as well as facilitate analysis of websites' effectiveness to sell products and create new customers.

Oracle WebCenter Sites: Engage lets you design online sites that gather information about your site visitors and customers. Marketing uses this information to personalize product placements and create promotional offerings for each visitor.

The Oracle WebCenter Sites: A/B Testing module provides a feature to compare two or more versions of a web page to determine the most effective version that can help converting a website visitor into a website customer through a sale. WebCenter Sites allows many methods to analyze effectiveness of a page version. Some of them are: visitor clicking a link, visiting a certain set of pages, remaining on the site for a certain length of time, adding an item to a shopping cart, and other actions, as well as the sale of a product. For information, see Developing for A/B Testing.

For detailed information, see Developing Personalized and Targeted Websites with Engage.

1.5.3 Creating Mobile Websites

The WebCenter Sites' mobility feature lets you easily extend your web presence to mobile devices and deliver multi-channel marketing and customer experience initiatives while saving significant time, money, and effort in managing mobile sites. Use the mobility feature to create, preview, and deliver websites to a variety of mobile devices such as phones and tablets. For information about developing mobile websites, see Developing Mobile Websites.

2

Overview of the Avisports Sample Site

Avisports, a sports-centric sample site, illustrates features such as creating and editing assets in Form Mode and Web Mode in the Oracle WebCenter Sites: Contributor interface. It includes articles illustrated with images. Avisports provides you with sample templates that are coded to render assets' Create and Edit view in Web Mode.

Topics:

- Touring the Avisports Sample Site as a Content Contributor

- Touring the Infrastructure of the Avisports Sample Site

2.1 Touring the Avisports Sample Site as a Content Contributor

To get a glimpse of what you can do with Oracle WebCenter Sites, take a tour of the avisports demo site. We used WebCenter Sites tools and technologies to design this site for you. By default, the Contributor interface is launched for you when you log into the WebCenter Sites instance.
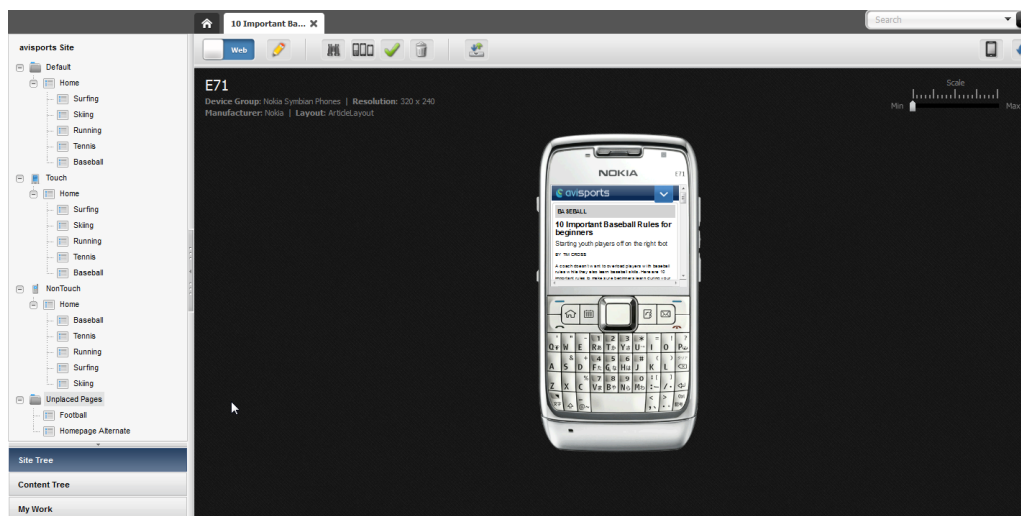
On the left of the Contributor interface is the **Site Tree** that contains three site navigations for the avisports site. The avisports site under the Default site navigation node is meant for desktop and laptop computers. The Touch site navigation node is meant for touch screen devices, and the NonTouch site navigation node is for non-touch devices with QWERTY keypads. Expanding these nodes displays the site pages. Double-clicking a page displays the page in Web Mode, where a content contributor can edit and preview the content.

You can preview the avisports site in the context of a single or multiple devices. To preview avisports in the context of a single device, click the **Show/Hide Devices** icon on the right and choose a device in which you wish to preview this site. For multi-

device preview, click the **Multi-Device Preview** icon. These previews show how the site will be displayed on site visitors' devices.

See *Oracle Fusion Middleware Using Oracle WebCenter Sites*.

2.2 Touring the Infrastructure of the Avisports Sample Site

Now let's take a look at the infrastructural elements of the avisports site. To take a tour of the avisports infrastructure, you need to switch to the Admin interface. The **Admin** icon on the top left of the WebCenter Sites interface can quickly take you there.

In the Admin interface, click **Site** to display its contents. Expand the **Site Navigation** node. There are three nodes that contain assets identical to those on the **Site Tree** in the Contributor interface. Expand any site navigation node (Default, Touch, NonTouch) to see that it contains the Home page, navigation pages, articles, and images.
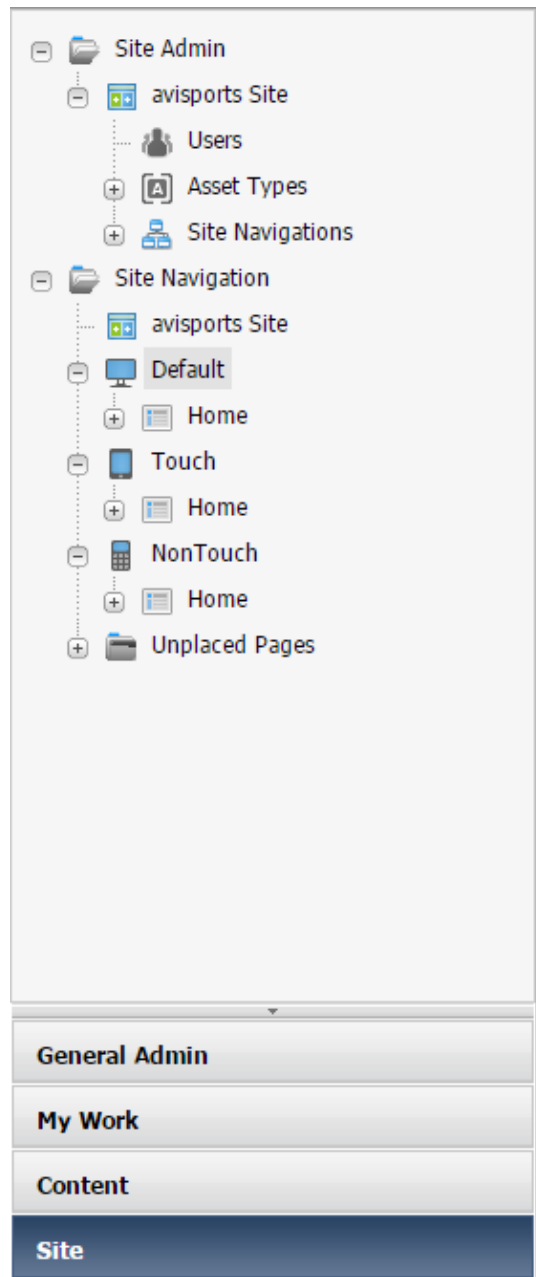
Expand the **Site Admin** node, and then expand the **Asset Type** node to view the assets enabled for the avisports site.

Expand the **Site Navigations** node. It lets you create new site navigations and reorder the existing site navigations.

On the **General Admin** tree, expand the **Mobility** node. It includes **Device Groups**, **Devices**, **Device Repository**, and **Image Properties** nodes. These nodes were used while designing avisports site for touch screen and QWERTY devices.

To view templates used for the avisports site, click **Search** on the toolbar. In the Search dropdown list, choose **Suffix** and select the default **All** in the **for** dropdown list. These choices display templates used for avisports meant for desktop and laptop devices, touch screen devices, and non-touch QWERTY devices.

What you are seeing in the Admin interface is the infrastructure of the site that you previewed in the Contributor interface.

To familiarize yourself with the high-level development process, see The WebCenter Sites Development Process. To learn what types of websites you can develop with WebCenter Sites, see Use Case Scenarios for WebCenter Sites.

3

The WebCenter Sites Development Process

In WebCenter Sites, you design two sites: a content management site(s) that the content contributors use to add and update information, and an online site that is delivered to visitors' browsers from your delivery system. Content contributors publish the online site from the management system to the delivery system.

So you are responsible for the user experience of two sets of end users:

- The site visitors who use your delivery system.

- The content providers who use the management system.

When creating these two closely connected yet separate sites, the development team performs a series of planning, development, and testing steps. This chapter describes the development process in one possible sequence of events and in very general terms. Your own workflow will vary based on your work environment and business needs.

Topics:

- Step 1: Set Up the Team

- Step 2: Create Functional and Design Specifications

- Step 3: Set Management System Requirements

- Step 4: Implement the Data Design

- Step 5: Build the Online Site

- Step 6: Set Up the Management System

- Step 7: Set Up the Delivery System

- Step 8: Publish to the Delivery System

3.1 Step 1: Set Up the Team

People with a variety of skillsets collaborate to assemble a site in WebCenter Sites. Some people design the site, some code various elements, some manage content, and so on.

- Solution architects (site designers)

- XML and JSP developers

- Java application developers

- Database administrators

- System network administrators

- Marketers and advertising staff

- Product managers (if you are developing a commerce site)

- Content providers

You need people such as DBAs, system administrators, and content providers on your development team in addition to the people (like you) who do the actual coding for several reasons:

- You need to design a data model in addition to creating a page design, which means that you need early input from the DBAs who will be supporting the databases on each system.

- Code and data need to move around on multiple separate systems, several of which are probably clustered, which means you need early input from system and network administrators.

- Implementing a WebCenter Sites system is dependant on the work habits of your content providers being accurately reflected in the design of the management system. You need early input from those who will use the management system.

3.2 Step 2: Create Functional and Design Specifications

An online site delivered from an Oracle WebCenter Sites content management system is a holistic construct in which everything interacts, intersects, and works with everything else. So, you need to create a functional specification and a design specification (to design your online site on paper).

You should accomplish some pieces of this task before you begin coding anything (although you might do some proof-of-concept coding while working on the design specification).

3.2.1 Functional Requirements

Before you can begin a design specification, product management and marketing must provide the functional requirements for the online site.

3.2.2 Page Design

After you obtain the functional requirements from your marketing folks, a good place to start is to map out all the types of pages that you want to present on the online site. For example, home page, section page, columnist page, search page, article page, and so on. To design a commerce site you will need other kinds of pages: registration page, product category pages, product description page, article page, FAQ page, invoice page, and so on.

Determine the graphical, navigational, and functional features for each page and the site overall: navigation bars, buy buttons and shopping carts, tell me more buttons, search functions, logo placement, animated graphics, and so on.

If you are using Oracle WebCenter Sites: Engage, decide where the merchandising messages (recommendations) are to be placed on the pages and on which pages they'll be placed. For example, each product category page may include a New Products section in the upper-right corner of the page.

Map out the entire structure of the site and create mock-ups.

3.2.3 Caching Strategy

One of the major elements in your design is caching: page caching and resultset caching. No online site can reach performance goals without you planning, testing, and implementing a caching strategy. While designing the pages that you want to present on your online site, you must consider how and when page caching can and should be implemented for each piece on each page. While designing your queries, you must map out all the tables in the database and determine how the resultset caching settings should be set for each table.

3.2.4 Security Strategy (Access Control)

You must determine what kinds of access control you want to enforce early in the design process so that you design your pages correctly. For example, requiring your visitors to identify themselves before they are allowed to access any part of your online site. The requirement to check visitors' identities before allowing them access to a page affects how you would cache the components of that page. You could design a container page, which is never cached. This page verifies the identity of the visitor and then assembles the page from cached pagelets only if the verification is successful.

3.2.5 Separate Format from Content (Elements from Assets)

Following the basic proposition of separating content from format, take a look at each piece of each proposed page in your site and determine whether that piece should be represented as data or as logic.

A good design is one in which data is designed to be represented as an asset and is not embedded into element code. Examine every component of design or content, and then determine what your assets are. You make that determination by deciding which category a component belongs to: data or logic/code.

Simply speaking, do not code something into an element (embed it in logic) if it is really data. Data should be in a separate asset.

Here's another way to look at it:

- Assets that represent content are the responsibility of content providers.

- Logic, anything coded into any element, is the responsibility of the developers.

3.2.5.1 Determine the Asset Types (Content)

Documents, articles, products, and images are easily identified as assets. However, design components such as headers and footers could also be assets:

- When the content in a header or footer is embedded in the code of an element, you or another developer has to change the text in it when anything in it changes (a phone number, a logo, and so on).

- When the content in a header or footer is in an asset, the code in your elements must be able to obtain the identity of the asset. Its content becomes the responsibility of a content provider.

Other page components that can be assets include the following:

- Animation and other media

- Quote of the day

- Company or stock profiles

- Knowledgebase questions and answers

From your point of view, someone else is responsible for a component's content which is represented in an asset. You are only responsible for when and where it displays on your online sites and what it looks like when it displays there.

3.2.5.2 Decide How to Handle Images and Other Blobs

You have two general options when deciding how to manage the images and other blobs that you want to use in your online site:

- Treat them as assets: Store them in the WebCenter Sites database and have the BlobServer servlet serve them.

- Treat them as static files: Put them in a file structure on your web server and let the web server serve them.

Either method is a valid option. You can create links to image files stored on the web server with the WebCenter Sites tags. There may be performance benefits when you allow your web server to deliver your images. However, if you keep your images and blobs separate from the WebCenter Sites database:

- You must implement a separate file management process. The publishing methods that move image assets from your management system to your delivery system cannot move content that is not in the WebCenter Sites database. You must manage this process on your own.

- None of the native WebCenter Sites security mechanisms will apply. That is, you cannot use ACLs to limit access to blobs that are not managed by WebCenter Sites.

3.2.5.3 Map Out the Functional Design and Format (Elements)

Analyze all of the functionality that you plan to incorporate into your online site. Parts of a commerce site will no doubt behave more like an application. Outline what code or logic is required for your visitor registration pages, visitor data collection pages, shopping carts, personalization, and so on.

Remember that your WebCenter Sites system provides you with coding options: Java, XML, and JSP. As you look at each of the functions you want to provide, determine which is the best coding solution for that function.

3.2.6 Data Design

Once you know which pieces of your site should be represented as assets, you can map out what your asset types should be. Each new asset type will use one or more database tables (depending on whether it is a basic or flex asset type).

3.2.6.1 Asset Types

No matter which asset model you are using, basic or flex, consider the following when you design your asset types:

- Asset type design affects both of the user groups that you are designing for (visitors to the online site and the content providers who enter the data).

- Which types of assets have to be linked or related to other assets of other types to successfully implement your page design? Be sure to implement these relationships in the asset type.

- Be sure that your asset types store only the data that you really plan to use so that content providers do not waste time maintaining data that no one uses.

3.2.6.2 Auxiliary Tables That Support Your Asset Types

The data design that you want to implement for your system extends beyond the database tables that hold your assets. Depending on the kinds of information that you want to provide, you might have to create auxiliary tables that support your asset types. For example, in a site that has asset types with a **Mimetype** drop-down list, a user must select a value from this list. You could create a lookup table named `MimeType` and pull these values from this table. Depending on your needs, you might have to create similar tables for your system.

Your DBAs should be involved in your discussions about the asset types and auxiliary tables that you plan to create so they can understand from the start the kind of database tuning issues that might arise on the management and delivery systems.

3.2.6.3 Visitor Data

If you are using Engage, determine what kinds of visitor data will be gathered. These data types are represented by the Engage visitor data assets that you use to create segments for personalizing your site based on the identity of the visitor. For example, demographics, purchase history, or clickstream information.

After your WebCenter Sites system goes live and starts collecting visitor data, the tables that store that data grow very quickly. This is another area that you have to consult your DBAs about.

3.3 Step 3: Set Management System Requirements

Don't yet begin coding. Think about how the management system should be organized. Keep in mind that the design depends on the content management site.

A content management site is an object that you use as an organizational construct for an actual online site and as an access control tool. When you create Template assets, WebCenter Sites creates an entry in the `SiteCatalog` table. The naming convention for the page entries includes the name of the content management site that you are creating the Template for. This means that you must be consistent with site names throughout your entire content management system (development system, management system, and delivery system), and you must know the names of the sites that you are using before you begin coding.

Although your primary concern is the name of each site, the system administrators and business managers must also determine the following:

- How many users and ACLs (access control lists) do you need? (Remember that you may have to create ACLs to assign to the visitors of the online site, as well.)

- How many site roles you do you need?

- Which asset types need a workflow process?

- Which asset types should use revision tracking?

- Who should have access to which asset types on which sites?

Use both this guide and *Oracle Fusion Middleware Administering Oracle WebCenter Sites* to help you make these decisions.

3.4 Step 4: Implement the Data Design

You've created the design specification, and you understand how the management system is organized. So, it's time for you to implement the data design.

On the development system, you complete tasks such as these:

- Create content management sites with the same names as those that will be used on the management system. See Creating a Site From the Admin Interface in *Oracle Fusion Middleware Administering Oracle WebCenter Sites*.

- Design and create your asset types. See Creating Basic Asset Types and Creating a Flex Asset Family.

- Add any lookup tables or other auxiliary tables for the asset types.

- Create sample assets of each type. See Creating an Asset in Form View in *Oracle Fusion Middleware Using Oracle WebCenter Sites*.

This step and Step 5: Build the Online Site are iterative and will most likely overlap a great deal. While you create asset types so that you can create assets before you create templates for them, it is likely that you will uncover areas that need refinement in your data design only after you have coded a template and tested the code.

3.5 Step 5: Build the Online Site

As soon as you've created your design specification, you can begin coding elements that do not display assets. And, to start coding templates and building the online site all you need is the sample assets of one type.

In this step, you complete tasks such as these:

- Create the page, query, and collection assets that implement the functionality of your online site. See Creating Collection Assets, Query Assets, and Page Assets.

- If you are using Engage, create the visitor data assets, sample segments, recommendations, and sample promotions. See Developing Personalized and Targeted Websites with Engage.

- Create Template assets (and code template elements) for all of your asset types. See Creating Template, CSElement, and SiteEntry Assets.

- For the Web Mode feature of the Oracle WebCenter Sites: Contributor interface, code the templates using the `insite` family of tags. See Coding Templates for In-Context Content Editing.

- Code the CSElements that implement underlying functionality (that do not display assets). See Creating CSElement Assets.

- For a commerce site, code pages that implement the shopping cart. See What You May Need to Know About Shopping Carts and Engage.

- If you are using Engage, code pages that collect visitor data. See Collection of Visitor Data.

- Test everything.

  Perform both usability and market testing for your online site.

See Website Development with Tag Technologies.

3.6 Step 6: Set Up the Management System

After your online site becomes functional on the development system, it's ready to move to the management system.

The developers complete the following task:

- Publish the content management site and all its components to the management system. See Working with RealTime Publishing in *Oracle Fusion Middleware Administering Oracle WebCenter Sites*.

The system administrators then complete the following kinds of tasks:

- Create users, ACLs, and roles. Assign users their roles for each content management site. See Working with ACLs and Roles in *Oracle Fusion Middleware Administering Oracle WebCenter Sites*.

- Create workflow processes. See Creating and Managing Workflow Processes in *Oracle Fusion Middleware Administering Oracle WebCenter Sites*.

- Create StartMenu shortcuts. See Creating a Start Menu Item in *Oracle Fusion Middleware Administering Oracle WebCenter Sites*.

- Enable revision tracking. See Enabling Revision Tracking in *Oracle Fusion Middleware Administering Oracle WebCenter Sites*.

For information about setting up the management system, see *Oracle Fusion Middleware Administering Oracle WebCenter Sites*.

3.6.1 Import Content as Assets

It is likely that you have content in some non-asset format that you want to use. To import this content into the WebCenter Sites database as assets, use the XMLPost utility. See About Importing Assets Using the XMLPost Utility.

3.6.2 Import Catalog Data and Flex Asset Data

For the flex asset model, you can import a large amount of pre-existing data with the BulkLoader utility. See Importing Flex Assets with the BulkLoader Utility. For systematic updates, however, you use the XMLPost utility. See Using the XMLPost Utility.

3.6.3 Instruct the Editorial Team About Site Design

Before the editorial team can successfully maintain the online site, they must understand your design. For example, how frequently are collections supposed to be rebuilt?

If you are using the basic asset model, content providers have to know the following:

- Which categories and sources they should assign to their assets in order for their assets to be located by the appropriate queries and collections.

- Which templates they should assign to which assets.

- Which association fields must be filled out in order for the links on the site pages to function correctly.

It is a good idea to program as much of this information as possible into the start menu shortcuts that you and the system administrators create for each asset type.

If you are using the flex asset model, content providers have to know the following:

- The general hierarchy or taxonomy in place for the flex assets.

- Some information about what information a flex asset inherits.

- Which templates they should assign to which assets.

3.7 Step 7: Set Up the Delivery System

You're ready to publish all assets on the management system to the delivery system as soon as you've set up the delivery system. Since the delivery system hosts the public-facing site, you don't have to publish Start menus, workflows, revision tracking, and so on which you configured on the management system for content contributors and marketers.

On your delivery system, you need to also accomplish these tasks:

- Implement your security strategy. See Setting Up External Security in *Oracle Fusion Middleware Administering Oracle WebCenter Sites*.

- On the web server, map the URL of your site (`www.example.com`) to the WebCenter Sites URL of your home page. See Mapping a URL Prefix for Your Web Server in *Oracle Fusion Middleware Administering Oracle WebCenter Sites*.

For information about setting up the delivery system, see the section on publishing in *Oracle Fusion Middleware Administering Oracle WebCenter Sites*.

3.8 Step 8: Publish to the Delivery System

Your content is published to the delivery system. Open your site to the public only after an intensive testing (both performance and load).

See Working with RealTime Publishing in *Oracle Fusion Middleware Administering Oracle WebCenter Sites*.

Part II

Building Your Data Model

Before you start developing a site, get an insight into designing basic and flex asset types. Also become familiar with what a flex filter class is, how you would create a flex filter asset and a flex family, and how you would design attribute editors and configure their instances. Learn about various kinds of tables and columns in the WebCenter Sites database, how you can create database tables, and interact with those tables that do not hold assets.

- Understanding the Asset Types and Asset Models

- Designing Basic Asset Types

- Designing Flex Asset Types

- Creating a Hierarchical Flex Family

- Creating Flex Filters

- Designing Attribute Editors

- Configuring Bundled Attribute Editors

- Working with the WebCenter Sites Database

- Managing Data in Non-Asset Tables

4

Understanding the Asset Types and Asset Models

You need basic and flex asset models to develop sites. The basic model comprises simple, unalterable asset types where each type is stored in a single table. The flex model includes a flex family whose asset types inherit attributes that are changeable and are stored across multiple tables.

Topics:

- What Are Asset Types?

- What Are Asset Models?

- The Basic Asset Model

- The Flex Asset Model

- Assetsets and Searchstates

- Search Engines and the Two Asset Models

- Tags and the Two Asset Models

- Summary: Basic and Flex Asset Models

> **Note:**
>
> Designing and creating tables that do not hold assets is discussed in Working with the WebCenter Sites Database.

4.1 What Are Asset Types?

Content contributors and marketers can create assets such as articles, images, videos when you make asset types for those assets available to them. When creating your asset model, you may want to leverage the asset types available in the avisports sample site.

An *asset type* defines the characteristics of asset objects of that type. An *asset* is an object that is stored in the WebCenter Sites database and can be created, edited, inspected, deleted, duplicated, placed into workflow, tracked through revision tracking, searched for, and published to your delivery (live) site.

This figure shows the WebCenter Sites Content-Entry form and the relationship of field names and field values to the database table for an asset type.

Figure 4-1   Asset Types: Database Tables and WebCenter Sites Forms

WebCenter Sites Content-Entry Form

Spark Contact: (Contact)

*Name: John Doe
Description:
Filename:
Path:
External Item ID:
Spark Content Definition: Contact
Phone: 516-555-5555
Email: johndoe@retail.com
Ratings: [no Segments defined]
                     no segment ratings apply
Related Items: [no recommendations defined]

Cancel   Save

Field names define the asset type

Name	Phone	Email	Ratings
John Doe	516-555-5555	johndoe@retail.com	

Field values define the asset

Database Table for Asset Type "Contact"

Developers design and create asset types while designing your content management system and your online sites. Content providers then create and edit assets of those types.

In general, assets perform one of the following three roles:

- Provide content that visitors read and examine on your online sites

- Provide the formatting logic or code for displaying the content

- Provide data structure for storing the content in the WebCenter Sites database

The developer's job is to design asset types that are easy for content providers to work with on the management system and that can be delivered efficiently to visitors from the delivery system.

Several core asset types are delivered by WebCenter Sites and Oracle WebCenter Sites: Engage. Because WebCenter Sites has a stack architecture, the core asset types are made available as follows:

- WebCenter Sites delivers the template, query, collection, SiteEntry, CSElement, Link, and page asset types. All of the other modules and products use the template and page asset types.

- WebCenter Sites delivers the attribute editor asset type. It supports any flex attribute asset types that you create.

- Engage delivers the visitor attribute, history attribute, history definition, segment, recommendation, and promotion asset types.

Assets of these types provide format or logic for the display of asset types that hold your content by retrieving, ordering, organizing, and formatting those assets. In other words, you use the core asset types to organize and format the content on your online site.

4.1.1 Asset Types Delivered with WebCenter Sites

Asset types delivered with WebCenter Sites provide basic site design logic. You can create as many individual assets of these types as you need, but you cannot modify the asset types themselves:

- **Query**: Stores queries that retrieve a list of assets based on selected parameters or criteria. You use query assets in page assets, collections, and recommendations. The database query can be either written directly in the New or Edit form for the query asset as a SQL query, or written in an element (with WebCenter Sites query tags or a as a search engine query) that is identified in the New or Edit form.

- **Collection**: Stores an ordered list of assets of one type. You build collections by running one or more queries, selecting items from their resultsets, and then ranking (ordering) the items that you selected. This ranked, ordered list is the collection. For example, you could rank a collection of articles about politics so that the article about last night's election results is number one.

- **Page**: Stores references to other assets. Arranging and designing page assets is how you represent the organization or design of your site. You design page assets by selecting the appropriate collections, articles, imagefiles, queries, and so on for them. Then, you position your page assets on the **Site Navigation** node that represents your site in the tree on the left side of the WebCenter Sites Admin interface.

  Note that a page asset and a WebCenter Sites page are quite different. The page asset is an organizational construct that you use in the **Site Navigation** node as a site design aid and to identify data in your elements. A WebCenter Sites page is a rendered page that is displayed in a browser or by some other mechanism.

- **Template**: Stores code (XML or JSP and Java) that renders other assets into WebCenter Sites pages and pagelets. Developers code a standard set of templates for each asset type (other than CSElement and SiteEntry) so that all assets of the same type are formatted in the same way.

  Content providers can select templates for previewing their content assets without having access to the code itself or being required to code.

- **CSElement**: Stores code (XML or JSP and Java) that does not render assets. Typically, you use CSElements for common code that you want to call from multiple templates (a banner perhaps). You also use CSElements to provide the queries that are needed to create DynamicList recommendations in Engage.

- **SiteEntry**: Represents a WebCenter Sites page or pagelet and has a CSElement assigned as the root element that generates the page. Template assets do not have associated SiteEntry assets because they represent both an element and a WebCenter Sites page.

- **Link**: Stores a URL to an external website. You use this asset to embed an external link within another asset.

- **Attribute Editor**: Is an attribute editor that specifies how data is entered for a flex attribute when that attribute is displayed on a New or Edit form for a flex asset or a flex parent asset. It is similar to a Template asset. However, unlike a Template asset, you use it to identify the code that you want WebCenter Sites to use when it displays an attribute in its interface, not when it displays the value of an attribute on your online site.

Because the data needs of each organization using a WebCenter Sites content management system are different, there are no default asset types that represent content. However, the sample sites deliver sample content asset types that you can examine and modify for use on your sites.

4.1.2 Asset Types Delivered with Engage

The Engage application delivers several core asset types that you use to gather visitor information so that you can personalize the product placements and promotional offerings that are displayed for each visitor:

- **Visitor Attribute**: Holds types of information that specify one characteristic only (scalar values). For example, you can create visitor attributes named `years of experience`, `job title`, or `number of children`.

- **History Attributes**: Are individual information types that you group together to create a vector of information that Engage treats as a single record. This vector of data is the history definition. For example, a history type called `purchases` can consist of the history attributes `SKU`, `itemname`, `quantity`, and `price`.

- **Segments**: Are assets that divide visitors into groups based on common characteristics (visitor attributes and history types). You build segments by determining which visitor data assets to base them on and then setting qualifying values for those criteria. For example, a segment could define people who live in Alaska and own fly fishing gear, or it could define people who bought a personal computer in the past six months, and so on.

After you define and categorize the visitor data that you want to collect, you use the following asset types to select, organize, and display the flex assets that represent your content on your online site:

- **Recommendation**: Is like an advanced collection. It collects, assesses, and sorts flex assets (products or articles, perhaps) and then recommends the most appropriate ones for the current visitor, based on the segments that visitor belongs to.

- **Promotion**: Is a merchandising asset that offers some type of value or discount to your site visitors based on the flex assets (products, perhaps) that the visitor is buying and the segments that the visitor qualifies for.

> **Note:**
>
> Engage interacts with assets that are built using the flex asset model only. You cannot program recommendations and promotions to work with assets that use the basic asset model.

4.2 What Are Asset Models?

Asset models (or data models) consist of asset types that content contributors and marketers use to create content assets. You can design two types of models, basic and flex. The asset types in the basic model are unalterable and suitable for predictable scenarios. The asset types in the flex model are hierarchal and changeable.

- **Basic**: Asset types have a simple data structure. They have one primary storage table and simple parent-child relationships with each other.

  Basic asset types are separate, standalone asset types that represent individual kinds of content: an article, an image file, a page, a query, and so on. You use the AssetMaker utility (located in the **General Admin** tree under the **Admin** node) to create basic asset types.

- **Flex**: Asset types have a complex data structure with several database tables and the ability to support many more fields than do basic asset types. Additionally, they can have multiple parents, any number of grandparents, and so on, that they can inherit attribute values from.

  Flex asset types comprise families of asset types that define each other and assign attribute values to each other. You use the Flex Family Maker utility (located in the **General Admin** tree under the **Admin** node) to create a family of flex asset types.

During the process of designing your online site with the WebCenter Sites content management system, you and others on your team create the asset types that should represent the content for your site. The WebCenter Sites template and page asset types provide the formatting framework for the asset types that represent your data, whether you use the basic data model or the flex data model.

The asset data model (basic or flex) that you should choose to represent the data that you want to display on your online site depends on the nature of that data, as described in the following two sections:

- When to Use the Basic Model

- When to Use the Flex Model

4.2.1 When to Use the Basic Model

The basic model is a good choice when your data has the following characteristics:

- It is fixed, predictable: there will be no need to add attributes to the asset type.

- It is homogenous: all assets of the same type have similar attributes.

- It has a moderate number of attributes. You are limited by your database as to how many columns/attributes you can have in the asset type table for a basic asset.

- You want to use the static publishing method. There are very limited applications of the flex asset model in which it makes sense to use the static publishing method.

- Visitors browse your online site by navigating from link to link.

When the data for an asset type can be imagined as a spreadsheet, as a simple flat table where each asset of that type is a single record and every record has the same columns, that asset type should use the basic asset model.

4.2.2 When to Use the Flex Model

The flex model is the right choice when your data has the following characteristics:

- It has lots of attributes. For example, products can have potentially hundreds of attributes. Because attribute values for the flex family member are stored as rows rather than columns, and flex assets can physically have many more attributes than basic assets can.

- It can be represented in a hierarchy in which assets inherit attribute values from parent assets.

- You cannot predict what attributes might be necessary in the future and your data might need additional attributes periodically.

- Asset instances of the same type can vary widely. That is, not all assets of that type should have the same attributes. For example, a bath towel product asset would have attributes that a toaster product asset would not, but both the bath towel and the toaster are product assets.

- Visitors browse your online site by navigating through drill-down searches that are based on the attribute values of your data.

- You want to use Engage.

Products fit into the flex asset model because markets are constantly changing. You cannot always predict what products you will be selling next year or what attributes those products will have.

Flex data model is the right fit when you business needs require you to make modifications to your asset types such as adding or changing their attributes. The flex asset model gives you the extensibility that you have to represent data whose characteristics cannot be predicted.

4.3 The Basic Asset Model

WebCenter Sites includes the basic asset model by default. This data model uses one database table per asset type. All basic assets of the same type have the exact same fields (properties), and all assets of a single type are stored in the same database table. Most of the core WebCenter Sites asset types use this model.

The AssetMaker utility lets you create basic asset types. You code XML files called asset descriptor files using a custom tag named PROPERTY and then upload the file with AssetMaker. A property is both a column and a field. A PROPERTY statement defines a column in the table that stores assets of that type and defines how data is to be entered into the corresponding field for that column in the WebCenter Sites forms.

For information about coding asset descriptor files and creating new basic asset types, see Designing Basic Asset Types.

Familiarize yourself with the following:

- Relationships Between Basic Assets

- Category, Source, and Subtype

- Basic Asset Types and the Database

4.3.1 Relationships Between Basic Assets

Basic asset types have very simple parent-child relationships. You use these relationships to associate or link assets to each other. When you design the online pages for your online sites you code template elements that identify, extract, and then display an asset's children or parent assets in appropriate ways.

The relationships that basic assets can have with each other are called *associations* and *unnamed relationships*. When these relationships occur between individual assets, they are written to the `AssetRelationTree` table.

4.3.1.1 Associations

Associations are defined, asset-type-specific relationships that are represented as fields in the WebCenter Sites asset forms. After you create an asset type with AssetMaker, you use the Association form for that asset type to create association fields.

You use associations to set up relationships that make sense for the asset types in your system and then you use the names of these relationships to identify the related assets and display them in appropriate ways on your site pages. For example, in a site where an asset named article had three associations with an imagefile asset type, such as Main ImageFile, Teaser ImageFile, and Spot ImageFile, the article templates would be coded to display the imagefiles that are linked to articles through these associations. The association is what enables the template to determine which imagefile is the correct one to display for an individual article asset. When a content provider selects an image asset in the imagefile field of the New and Edit article forms, the selected imagefile asset becomes a child of the article asset. (Note that this same imagefile asset can also be a child of other articles.)

When you create a new association between asset types, WebCenter Sites creates a row for that type of association in the `Association` table. Then, when you create an asset and specify the name of another asset in an association field, that relationship is written to the `AssetRelationTree` table.

4.3.1.2 Unnamed Relationships

Unnamed relationships occur when you build a collection, the items in the collection become children of the collection.

4.3.2 Category, Source, and Subtype

There are three additional ways to organize or categorize basic assets: category, source, and subtype. Categories and subtypes are specific to an asset type. Source, however, applies to all the asset types in a content management site. In other words, source is site-specific.

4.3.2.1 Category

`Category` is a default column and field that you can use to categorize assets according to a convention that works for your sites. Although all basic asset types have a `category` column by default, you do not have to use it (not a required field).

For example, a banking site might have categories named Personal Finance, Banking and Loans, Rates and Bonds, News, and so on. Articles identified with these categories are selected by queries that use category as a selection criterion and displayed on specific site pages, as appropriate.

When you create a new basic asset type, AssetMaker creates one category code for assets of that type. You then use the Category form for your new asset type to create additional categories to use this feature.

New categories are written to the `Category` table, which serves as the lookup table for the **Category** field on the New and Edit asset forms for asset types that use the basic asset model.

The purpose of the **Category** field and column is for site design. You can use category, or not, in your queries and query assets for your online site. The WebCenter Sites application does not base any of its functions on category codes. (With the exception that you can search for assets based on this field, if you are using it.)

> **Note:**
>
> SQL Server can't differentiate case for attribute categories. For example, you can't have a category named `MyCategory` and another category named `mycategory` if you are using SQL server.

4.3.2.2 Source

`Source` is a column and field that you can use to identify where an asset originated. Although WebCenter Sites provides administrative support (through the Source form) for you to use this feature in the design of your online site, the `source` column does not exist by default in the primary storage tables for basic asset types other than `Article`. To use source with your basic asset types, you must include a property statement in your asset descriptor file for it.

For example, a banking site might have sources named WireFeed, Asia Pulse, UPI, and so on. Certain online pages select stories to display based on the results of queries that search for articles based on the value in their source column.

After you create a new basic asset type, you add new sources in the Source form in the **Admin** node on the **General Admin** tree, if necessary. New sources are written to the `Source` table, which serves as the lookup table for the **Source** field on the New and Edit asset forms for basic-style assets.

4.3.2.3 Subtype

The subtype concept provides a way to further classify an asset type. In the flex asset data model, the definition asset types create subtypes of flex assets and flex parent assets. In the basic asset data model, the concept of subtype is implemented through the `subtype` column in the primary storage table for the asset type.

The WebCenter Sites application uses the value of an asset's Subtype in many ways:

- For Template assets, subtype means the type of asset that the template formats. Templates that format articles are a different subtype of template than templates that format images. When you create an article asset, only the templates that format articles appear as options in the **Template** field on that asset's New or Edit form.

In addition, you can use the Oracle WebCenter Sites: Contributor interface to specify a subtype that is displayed using a given template. For example, if your website uses two subtypes of article asset, Sports and News, you can create a template that only displays articles with the Sports subtype.

- For query assets, subtype means the type of asset that the query returns. Query assets that return articles are a different subtype of query asset than those that return imagefiles.

- For collection assets, subtype means the type of asset that the collection holds. Collections that hold articles are a different subtype of collection asset than those that hold imagefiles.

- For the basic asset types that you design, subtype is designed to classify an asset based on how it is rendered. You can define a default template for each subtype of an asset type for each of your publishing targets.

You are required to create subtypes only when a different template should be assigned to assets of a specific type based on the publishing target for the asset.

The field named Subtype is displayed in those assets' New and Edit forms for whose asset types you create any subtypes. The drop-down list in the field displays all the possible subtypes for that asset type.

Note:

In the flex asset model, the definition asset types serve as subtypes. For example, the flex family in the avisports sample site has a definition named `Article`. This means that there is one subtype for article assets: the `Article` subtype.

For some asset types, the subtype is set implicitly and cannot be changed. Other asset types allow users to choose a subtype for the asset using the Contributor interface. Table 4-1 lists the WebCenter Sites asset types according to whether they have configurable subtypes.

Table 4-1   Implicit Subtypes vs. Configurable Subtypes

Implicit Subtypes	Configurable Subtypes
• All flex assets • Query assets • Collection assets • Template assets • Page assets	• All custom basic assets (made with AssetMaker) • Article assets • Image assets • Linkset assets • Recommendation assets • Link assets

For information about setting configurable subtypes, see Designing Basic Asset Types.

4.3.3 Basic Asset Types and the Database

Although there is one primary storage table for basic asset types, WebCenter Sites keeps other kinds of supporting information for basic assets in other tables. When you create a new asset of a basic type, WebCenter Sites writes to the following database tables:

- The primary database table that holds assets of its type. For example, each page asset has a row in the `Page` table and each article asset has a row in the `Article` table.

  These tables store all of the asset's attribute or field values, such as the asset's name, its object ID, who created it, which template it uses, and so on. The name of this table always matches the name of the asset type.

  When you create a new basic asset type, the AssetMaker utility creates the primary storage table (a WebCenter Sites object table) for the asset type as a part of that process.

- The `AssetRelationTree` table, if the asset has associations with other assets. The relationships that basic assets can have are described in Relationships Between Basic Assets.

- The `AssetPublication` table, which specifies which content management sites (publications) give you access to the asset. If the asset is shared among sites (publications), there is a row entry for each pubid. A *pubid* is a unique value that identifies a site (publication).

- The `SitePlanTree` table, if the asset is a page asset. This table stores information about the page asset's hierarchical position in your site navigation.

When you develop the templates that display the assets that represent your content, you code elements with XML or JSP tags that extract and display the information from the tables in the preceding list.

Be sure to examine the New and Edit forms for the various sample asset types and to use the Explorer tool to examine the tables in your WebCenter Sites database.

> **Note:**
>
> Do not use the Explorer tool to modify the data in any of these tables. All editing of assets and their related tables should be done only through the WebCenter Sites interface.

4.3.3.1 Template Asset Type and the Database

Although the Template asset type is a core asset type, it does not use the basic asset model. It is a complex asset type with entries in the following database tables:

- The `Template` table (primary storage table)

- The `SiteCatalog` table

- The `ElementCatalog` tables

When you create a new Template asset, WebCenter Sites automatically creates entries in both the `SiteCatalog` and `ElementCatalog` tables for it.

4.3.3.2 Default Columns in the Basic Asset Type Database Table

WebCenter Sites needs several default columns for its basic functionality. So, AssetMaker creates each of the columns described in the following table in the asset type's primary storage table in addition to the columns defined in the asset descriptor file for that asset type.

Note that you do not have to code your asset descriptor files to include property statements for the columns in this table:

Table 4-2   Columns in an Asset Type's Primary Storage Table

Default Column (Field) Name	Description	Where It's Displayed in the WebCenter Sites Interface
`id`	A unique ID for each asset, automatically generated by WebCenter Sites when you create the asset. You cannot change the value in this field.	Forms: • Inspect • Edit • Status • search forms
`name`	A unique name for the asset. Names are limited to 64 alphanumeric characters.	Forms: • New • Edit • Inspect, • Status Also in the search results lists.
`descripti` `on`	A short description of the asset that offers more information than just the name.	Forms: • New • Edit • Inspect • Status Also in the search results lists.
`status`	The status of the asset, one of the following status codes obtained from the `StatusCode` table: PL: created ED: edited RF: received (from XMLPost, for example) UP: upgraded from Xcelerate 2.x VO: deleted (void) WebCenter Sites controls the value in this field. This field cannot be edited manually.	Forms: Status, if the status of an asset is either PL (created) or ED (edited) Note that assets with a status of VO (deleted) are not displayed anywhere in the WebCenter Sites Windows interface.
`createdby`	The identity of the user who originally created the asset. This user name is obtained from the `SystemUsers` table. WebCenter Sites controls the value in this field. It cannot be edited manually.	Forms: Status Also, the Revision History list if revision tracking is enabled for assets of this type.

Table 4-2   (Cont.) Columns in an Asset Type's Primary Storage Table

Default Column (Field) Name	Description	Where It's Displayed in the WebCenter Sites Interface
createddate	The date and time that the asset was written to the database for the first time. WebCenter Sites controls the value in this field. It cannot be edited manually.	Forms: Status Also, the Revision History list if revision tracking is enabled for assets of this type.
updatedby	The identity of the user who most recently modified the asset in any way. This user name is obtained from the SystemUsers table. WebCenter Sites controls the value in this field. It cannot be edited manually.	Forms: Status Also, the Revision History list if revision tracking is enabled for assets of this type.
updateddate	The date on which the information in the status field was changed to its current state. WebCenter Sites controls the value in this field. It cannot be edited manually.	Forms: Status Also, the Revision History list if revision tracking is enabled for assets of this type.
startdate	Promotion assets (an Engage asset) have durations during which they can be displayed on the visitor pages on your live system. This column stores the start time of the promotion's duration. The promotion asset type is the only default asset type that uses this column. For information on using startdate and enddate fields for your asset types, see Example 5-7 in Creating Basic Asset Types.	Forms: • Duration, Edit, and Inspect for promotion assets. • New, Edit, Inspect, and Status if you enable it for other asset types.
enddate	For promotion assets (an Engage asset), this column stores the end time of the promotion's duration. The promotion asset type is the only default asset type that uses this column.	Forms: • Duration, Edit, and Inspect for promotion assets. • New, Edit, Inspect, and Status if you enable it for other asset types.

Table 4-2   (Cont.) Columns in an Asset Type's Primary Storage Table

Default Column (Field) Name	Description	Where It's Displayed in the WebCenter Sites Interface
subtype	The value of the asset's subtype. The subtype is set in different ways for different assets. See Subtype.	Forms: • New, and Edit for Template assets (**Asset Type** field). • New, and Edit for query assets (**Result of Query** field). • New, and Edit for any asset type that has subtypes configured for it. • Set Default Templates.
filename	The name to use for the file created for this asset during the Export to Disk publishing method. The page and article asset types are the only asset types that have this field enabled by default. For information on using the filename field for your asset types, see Example 5-7 in Creating Basic Asset Types.	Forms: • New and Edit for page and article assets, by default. • New and Edit for any other asset type that has the field enabled.
path	The directory path to use for exported page files that are generated from child assets of this asset when the Export to Disk publishing method renders that asset into a file. The page and article asset types are the only asset types that have this field enabled by default. For information on using the filename field for your asset types, see Example 5-7. in Creating Basic Asset Types. in Creating Basic Asset Types.	Forms: • New and Edit for page and article assets, by default. • New and Edit for any other asset type that has the field enabled.
template	The template for the asset. This is the template that is used to render the asset when it is either published with Export to Disk or rendered on a live dynamic delivery system. This template is also used to calculate the dependencies when the asset is approved for the Export to Disk publishing method, unless the asset type has subtypes and there is a default approval template assigned for the asset based on its subtype.	Forms: • New • Edit • Inspect • Status

Table 4-2   (Cont.) Columns in an Asset Type's Primary Storage Table

Default Column (Field) Name	Description	Where It's Displayed in the WebCenter Sites Interface
category	The category code of the category assigned to the asset, if any. If you decide to use the category field to organize assets, you add category codes in the Asset Types forms in the **Admin** node on the **General Admin** tree.	Forms: • New • Edit • Inspect • Status
urlexternaldoc Deprecated	If the asset was entered with the Sites Desktop interface rather than the WebCenter Sites interface, stores the external document that is the source for the asset. WebCenter Sites controls the value in this field. It cannot be edited manually.	Not applicable
externaldoctype Deprecated	The mimetype of the file held in the urlexternaldoc field. WebCenter Sites controls the value in this field. It cannot be edited manually.	Not applicable
urlexternaldocxml	Reserved for future use.	Not applicable

4.4 The Flex Asset Model

In the flex asset model asset types inherit attributes. You can modify this model to meet unpredictable content needs. For example, a car company keeps adding new models. And, all models share some features. Content for such a car site needs the flex asset model.

The main characteristics of the flex asset model are:

- Flex assets are defined by flex definitions. A flex definition is an asset type that determines which flex attributes make up an individual flex asset. Flex definitions create subtypes of the flex asset type.

- The definition asset types create subtypes of flex and flex parent assets, which allows individual instances of a flex asset or flex parent asset type to vary widely.

- Flex attributes are assets. The flex data model lets you add flex attributes to (or remove them from) existing flex asset types at any time.

- Flex filters can take the data from one flex attribute, transform or assess it in some way, and then store the results in another flex attribute when you save the flex asset. The resulting value from a flex filter action is called a derived attribute value. See Creating Flex Filters.

- Flex assets can inherit attribute values, even derived values, from their flex parents, which means that you can represent your data in hierarchies.

You do not create individual flex asset types as you do basic asset types; instead, you create a flex family of asset types.

See these topics:

- The Flex Family

- The Flex Family in the Avisports Sample Site

- Flex Attributes

- Flex Parents and Flex Parent Definitions

- Flex Assets and Flex Definition Assets

- Flex Families and the Database

4.4.1 The Flex Family

The flex asset data model can be thought of in terms of a family of asset types. There are six asset types in a flex family. Five are required, the sixth is optional, as indicated in Table 4-3.

Table 4-3   Flex Family Members

Flex Family Member	Number Per Family
flex attribute asset type	one
flex parent definition asset type	one or more
flex definition asset type	one or more
flex parent asset type	one or more
flex asset type	one or more
flex filter asset type	none or more

Whereas some asset types are used exclusively by developers to create the other asset types in the data model, the flex asset type is always used by the content providers to create assets of that type. (When necessary, authorized users can be given access to additional flex family members.)

To create a flex family, you access the Add New Flex Family form by double-clicking **Add New Family**, located in the **General Admin** tree on the **Admin** node under **Flex Family Maker**. under the **Flex Family Maker** node in the Admin interface. In the form, you name each of the asset types in the family. For example, a site has a flex family named product family. The flex asset is called the product asset, the flex attribute is called the product attribute, and so on.

The key member of a flex family is the *flex asset*. The flex asset is the unit of data that you extract from the database and display to the visitors of your online site (delivery system). All of the other members in the family contribute to the flex asset member in some way.

While the flex asset is the key, the *attributes* are the foundation of the flex asset model. An attribute is an individual component of information. For example, color, height, author, headline. You use attributes to define the flex assets and the *flex parents*. Flex assets inherit attribute values from their parents who inherit attribute values from their parents and so on.

You decide which attributes describe which flex assets and which flex parents by creating templates with the *flex definition* and *flex parent definition* asset types. Flex parents and their definitions implement the inheritance of attribute values.

Note that a flex parent or a flex asset cannot be defined by attributes of two types. A site might have two or more kinds of attributes. For example: product attributes and content attributes. A product asset (the flex asset member in the product flex family) can be defined by product attributes only. Its definition cannot include content attributes.

A *flex filter* enables you to configure some kind of action to take place on the value of an attribute and then save the results of the action when the flex asset is saved. For example, you can configure a filter that converts the text in a Word file into HTML code.

In summary, the flex asset member of a flex family is the reason for the family, the unit of content that you want to display. The other members of a flex family provide data structure for the flex asset. However, because all of the members in the family are assets, you can take advantage of the standard WebCenter Sites features like revision tracking, workflow, search, and so on.

4.4.1.1 Parent, Child, and Flex Assets

When you are using the flex asset data model, the phrase parent-child relationship refers to the relationship between a flex asset and its flex parent asset(s). This is a different parent-child relationship than the ones that basic assets have through asset associations.

Although it is possible for flex assets to have the kinds of parent-child relationships that basic assets do, it is unlikely for the following reasons:

- WebCenter Sites provides the `ASSETSET` and `SEARCHSTATE` tag families, which you use instead of the collection and query asset types to select the flex assets that you want to display. For more information about this tag family, see Assetsets and Searchstates.

- Flex assets have no need for associations. For example, to assign an image file to a flex asset like a product, you can create an attribute that identifies the image file and assign it to the definition for the flex asset.

4.4.2 The Flex Family in the Avisports Sample Site

In the avisports sample site there is one flex family that you can examine. To better understand the following descriptions of the sample flex asset types, examine some article and image assets in the Contributor interface as you read this section.

The Avisports Flex Family

The flex family in the avisports sample site provides the data structure for the articles and images shown on the avisports sample site. It creates an online sports news website.

These are the asset types in the avisports flex family:

- **ContentAttribute**: The flex attribute asset type used to define the attributes for the articles, images, and parents in the avisports sample site. For example, there are attributes named headline, subheadline, author, relatedImage, and so on.

- **ContentParentDef**: The flex parent definition asset type in the avisports sample site's flex family. It is used to create subtypes of parents. There is one: Category.

- **ContentDef**: The flex definition asset type in the avisports sample site's flex family. It is used to create a subtype for each definition: `Article`, `ArticleImage`, and `Image`. All of the `AVIArticle` assets in this site are defined by the `Article` definition. The `AVIImage` assets in this site are defined by either the `Image` or `ArticleImage` definition.

- **ImageCategory**: A flex parent that represents categories of images such as Running images, Skiing images, Baseball images, and so on.

- **ArticleCategory**: A flex parent that represents categories of articles such as Running Articles, Baseball Articles, Skiing Articles, and so on.

- **AVIImage**: A flex asset type that stores an uploaded image file. Image assets represent the images that are shown on the avisports website.

- **AVIArticle**: A flex asset type that stores the text of an article and information about it. It has attributes such as headline, byline, subheadline, body, and so on.

Notice that there are three content definitions in the avisports sample site's flex family, and only two content asset types. The Article definition is used by the `AVIArticle` asset type. The `Image` and `ArticleImage` definitions are both used by the `AVIImage` asset type.

4.4.3 Flex Attributes

Flex attributes are the foundation of the flex asset model. An attribute represents one unit of information. You use attribute assets to define flex assets and flex parents. They are then displayed as fields in the New and Edit forms for your flex assets and their parents.

An attribute is similar to a property for a basic asset. As does a property, an attribute defines the kind of data that can be stored in a column in a WebCenter Sites database table and describes a field in the forms. However, while a property defines one column in an asset type's database table, an attribute is an asset with database tables of its own.

This data structure (attributes as assets rather than columns) is a one of the main reasons why flex assets are so flexible.

Once again, a flex parent or a flex asset cannot be defined by attributes of two types. For example, in a site that has a product flex family and a content flex family, the product asset type can be defined by product attributes only. Its definition cannot include content attributes.

> **Note:**
>
> The data stored by flex attributes that you removed is not deleted from the database (queries continue to return this data).

4.4.3.1 Data Types for Attributes

The data types for your attributes are defined by the WebCenter Sites database properties located in the `wcs_properties.json` file, categorized under Core.

The following table lists the data types for flex attributes, the properties that define the data types, and the files where the properties are located.

Table 4-4   Data Types for Flex Attributes

Type	Property
date	cc.datetime
float	cc.double
integer	cc.integer
money	cc.money
string	cc.varchar
text	cc.bigtext
asset	cc.bigint
blob	cc.bigint

4.4.3.2 Default Input Styles for Attributes

When a flex attribute is displayed as a field on a New or Edit form, it has default input styles based on its data types. The following list presents the default input styles for flex attributes:

- Date: input boxes that look like the following:

   Figure 4-2   Date Attribute

- `Float`: A text field with decimal position enforced.

- `Integer`: A text field.

- `Money`: A text field with currency format enforced.

- `String`: A text field that accepts up to 255 characters.

- `Text`: A text box. The number of characters that it accepts depends on the database and database driver you are using.

- `Asset`: A drop-down list of all the assets of the type that was specified.

- `Blob`: A text field with a **Browse** button.

Instead of using the default input style, you can create an attribute editor and assign it to the attribute. Attribute editors are assets but they are also similar to the INPUTFORM statement in an asset descriptor file for a basic asset -- they specify how data is entered

into the attribute field. For more information about attribute editors, see Designing Attribute Editors.

4.4.3.3 Foreign Attributes

You can have flex attributes that are stored in foreign tables, that is, foreign attributes. They are subject to the following constraints:

- The foreign table must be registered with WebCenter Sites. That is, the foreign table must be identified to WebCenter Sites in the `SystemInfo` table.

- The foreign table must have a column that holds an identifier that uniquely identifies each row. The identifier must have fewer than 20 characters.

- The foreign table must have a column that is reserved for the attribute data value, which can be of any appropriate data type. For example, for a string type attribute, the data type must be appropriate for a string.

4.4.4 Flex Parents and Flex Parent Definitions

Flex parents and their flex parent definitions are organizational constructs that do two things:

- Implement the inheritance of attribute values. The parent definitions set up (describe) the rules of inheritance and the parents pass on attribute values to the flex assets according to those rules of inheritance.

- Determine the position of a flex asset on the tabs that display your assets in the WebCenter Sites interface. The hierarchy of the parents and the flex assets on the tabs in that tree are based on the hierarchy set up with the parent definitions.

Each parent asset type has its own set of attributes, as specified in its parent definition. The parent definition creates a form that you see in the WebCenter Sites interface.

> **Note:** While creating a parent definition, if you add the parent definition as its own parent, publishing errors may occur.

You use parents to organize or manage the flex assets by passing on attribute values that are standard and do not have to vary for each individual child asset of that parent.

Parent asset types affect how you and the content providers see and interact with the data within the WebCenter Sites interface.

For example, say there is a site named ProductSite that has two parent definitions: `Category` and `SubCategory`. Their sole purpose is to create structure on the **Content Tree** (in the Contributor interface).

When the product parent's definition is `Category`, the product parent is displayed at the top level on the **Content Tree**. When the product parent's definition is `SubCategory`, the product parent is displayed at the second level and it has a parent of its own, as shown in the following:

Figure 4-3   Product Node in the Content Tree
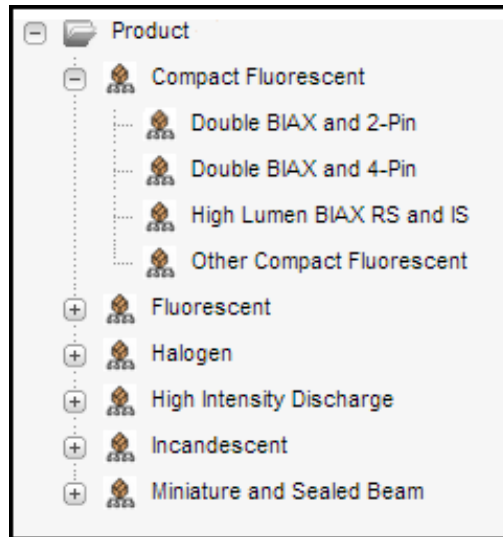
In Figure 4-3, there are several top-level product parents: Compact Fluorescent, Halogen, and so on. They were created with the Category definition. The next-level product parents, such as Double BIAX and 2-Pin, Double BIAX and 4-pin, and so on were created with the Subcategory definition.

4.4.4.1 Business Rules and Taxonomy

The purpose of parent definitions and parent assets is not only to express the taxonomy of your data. They also allow you to apply business rules (logic) without risk of input error from end users. If you created a flex asset of a specific definition and there are dependencies that it should inherit, that flex asset should have a parent.

For example, here is a simple product, a toaster with five attributes:

- SKU = 1234

- Description = toaster

- Price = 20

- CAT1 = Kitchen

- CAT2 = Appliances

When the value of CAT2 is Appliances, the value of CAT1 can only be Kitchen. In other words, there is a business rule dependency between the value of CAT1 and the value of CAT2.

In this kind of case, there is no reason to require the content providers to fill in both fields. Because every field whose data has to be entered manually is a field that might hold bad data through input error, you would use inheritance to impose the business rule:

- Make CAT1 and CAT2 parent definitions.

- Make Kitchen a parent created with the CAT1 definition and Appliances a parent created with the CAT2 definition.

- Make Kitchen the flex parent of Appliances.

Now, when content providers create products and select Appliances for CAT2, the value for CAT1 is determined automatically through inheritance.

4.4.5 Flex Assets and Flex Definition Assets

A *flex asset* is the reason for the flex family. It is the asset type that represents the end goal; a product, a piece of content that is displayed, and so on. For example, in the avisports sample site there are two flex asset types:

- `Article` (flex), an asset that holds text.

- `Image` (flex), an asset that holds an uploaded image file.

All of the other members in the family contribute to the flex asset member in some way.

A *flex definition asset* describes one kind of flex asset in a flex family; for example, a shoe, a toaster, a bowling ball, a brochure, a newsletter, an article, and so on. A flex definition asset is a template in that it directly affects a form that you see in the Contributor interface.

The avisports sample site has one definition for articles (`Article`) and two definitions for images (`Image` and `ArticleImage`). You can create as many flex definitions as you need. For example, a news site that should offer sports and weather articles will require a flex definition asset for sports and a different flex definition asset for weather.

Individual flex assets can be created according to only one flex definition asset. You could not create an image that used both the `Image` definition and the `ArticleImage` definition, but you can create an image that uses the `Image` definition and then create a different image that uses the `ArticleImage` definition.

A flex asset has not only the attributes assigned directly to it when it was created, it also has the attributes that it inherits from a parent. It can have multiple flex parents and whether the parents have parents depends on the hierarchical structure that you design. The following figure shows an example of a site that has three levels of hierarchy.

Figure 4-4   Levels of Hierarchy
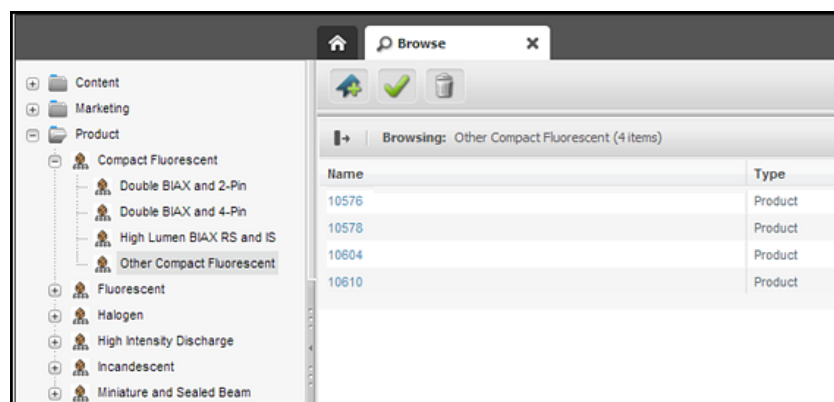
The Other Compact Fluorescent product parent has a parent of its own (Compact Fluorescent) and several children (10576, 10578, and so on).

4.4.6 Flex Families and the Database

Each asset type in a flex family has several database tables. For example, the flex asset member has six tables and a flex parent type has five. This data model enables the flex member in a flex family to support more fields than an asset type in the basic asset model can support.

The four most important types of tables in the flex model are as follows:

- The primary table for the asset type.

- The `_Mungo` table, which holds attribute values for flex assets and flex parent assets only.

- The `MungoBlobs` table, which holds the values of all the flex attributes of type `blob`.

- The `_AMap` table, which holds information about the inheritance of attribute values for flex asset and flex parents only.

There are several other tables that store supporting data about the relationships between the flex assets and additional configuration information (details about search engines, the location of foreign attributes, publishing information, and, if revision tracking is enabled, version information).

Additionally, certain kinds of site information are held in the same tables that basic assets use. For example, the `AssetPublication` table specifies which content management sites the asset type is enabled for.

When you develop the templates that display flex assets, you code elements that extract and display information from the `_Mungo` tables and the `MungoBlobs` table.

4.4.6.1 Default Columns in the Flex Asset Type Database Table

As do basic asset types, each of the flex asset types has a primary storage table that takes its name from the asset type. For example, the primary table for the avisports sample site asset type named article is `AVIArticle` and the primary table for this sample site's image asset type is `AVIImage`. The primary table for both article and image attributes is called `ContentAttribute`.

Unlike the primary table for a basic asset type, the primary table for a flex asset type has only the default columns. This is because flex asset types that have attribute values do not store those values in the primary table, attribute values are stored in the `_Mungo` table for the asset type.

In general, the default column types in the primary table for a flex asset type are the same as the default columns in the primary storage table for a basic asset type. For the general list of default column types, see Default Columns in the Basic Asset Type Database Table.

However, there are, of course, exceptions and additions, as described in the following table:

Table 4-5   Default Columns in the Flex Asset Type Database Table

Column	Description
category	Category is not used in the flex asset model so there is no `category` column in any of the primary tables for flex asset types. Flex assets have no need for the category feature because queries for flex assets are based on the values of their flex attributes.
template	Only the table for the flex asset member in a flex family, product, article (flex), and image (flex), for example, holds values in this column. This is because only the flex asset member in the family can have a Template asset assigned to it and be displayed on your online site.
renderid	Holds the object ID of the Template asset assigned to a flex asset.
attributetype e	An additional column in the primary table for flex attribute types. It holds the name of the attribute editor that formats the input style of the attribute when it is displayed in the New and Edit forms (if there is one).
flextemplate id	An additional column in the primary table for a flex asset type (the flex asset member of a flex family.) It holds the ID of the flex definition that the flex asset was created with.
flexgrouptem plateid	An additional column in the primary table for flex parent asset types. It holds the object ID of the parent definition that the flex parent asset was created with.

4.4.6.2 The _Mungo Tables

The flex asset and flex parent asset types have an `AssetType_Mungo` table, where `AssetType` is the name of the flex asset type (and matches the name of the main storage table). Its purpose is to store the attribute values assigned to an asset when an asset of this type is created. For example the avisports sample site table `AVIArticle_Mungo` holds the attribute values for article assets and the table `AVIImage_Mungo` holds the attribute values for the image assets.

Each attribute value has a separate row.

Each row in _Mungo table has a value in each of the columns described in Table 4-6.

Table 4-6   _Mungo Table Rows

Column	Description
id	A unique ID for each attribute value, automatically generated by WebCenter Sites when the flex asset is saved and the row is created. This is the table's primary key.
ownerid	The ID of the flex asset that the attribute value belongs to. (From the flex asset table: `Product`, for example.)
attrid	The ID of the attribute. (From the attribute table: `PAttributes`, for example.)
assetgroupid	If the attribute value is inherited, the ID of the parent who passed on the value. (From the parent table: `ProductGroups`, for example.)

Each row in a `_Mungo` table also has all of the columns in Table 4-7, but it has a value (data) in only one of them, depending on the data type of the attribute.

Table 4-7   _Mungo Table Columns

Column	Description
floatvalue	If the attribute's data type is `float`, the value of the attribute.
moneyvalue	If the attribute's data type is `money`, the value of the attribute.
textvalue	If the attribute's data type is `textvalue`, the value of the attribute.
datevalue	If the attribute's data type is `date`, the value of the attribute.
intvalue	If the attribute's data type is `int`, the value of the attribute.
blobvalue	If the attribute's data type is `blob`, the ID of the row in the `MungoBlobs` table that holds the value of the attribute.
urlvalue	If the attribute's data type is `url`, the path or url entered for the attribute.
assetvalue	If the attribute's data type is `asset`, the ID of the asset.
stringvalue	If the attribute's data type is `float`, the value of the attribute.

Because the `_Mungo` tables have URL columns (see Indirect Data Storage with the WebCenter Sites URL Field), a default storage directory (defdir) must be set for it. You use the `cc.urlattrpath` property in the `wcs_properties.json` file to set the defdir for your `_Mungo` tables.

4.4.6.3 The MungoBlobs Table

There is one `MungoBlobs` table. It holds all the values for all flex attributes of type `blob`, for all the flex attribute types in your system. Each attribute value has a separate row in the table.

4.4.6.4 The _AMap Tables

Flex asset and flex parent asset types have an `AssetType _AMap` table. Its purpose is to map the asset to the attributes it inherits from its parents. Then when you create a template that displays the asset on a page in your online site, you can query for assets based on any of their attributes and display any of those attributes, whether they were inherited or were directly assigned.

The `_AMap` table has one row for each flex asset that has a value for the inherited attribute. (However, if an attribute has multiple values, the `_Mungo` table has a row for each value.)

An `_AMap` table has the columns described in Table 4-8.

Table 4-8   _AMap Table Columns

Column	Description
id	A unique ID for each row, automatically generated by WebCenter Sites when the flex asset is saved and the row is created. This is the table's primary key.

Table 4-8   (Cont.) _AMap Table Columns

Column	Description
inherited	The ID of the parent the attribute was inherited from, if it was inherited. (From the parent table: ProductGroups, for example.)
attributeid	The ID of the attribute. (From the attribute table: PAttributes, for example)
ownerid	The ID of the flex asset that the attribute value belongs to. (From the flex asset table: Product, for example.)

4.5 Assetsets and Searchstates

`assetsets` display assets, and `searchstates` identify which flex assets should be displayed in an `assetset`.

Assetset

An *assetset* is a group of flex assets or flex parent assets *only*, not flex attributes or flex definitions or flex parent definitions. For example, in the avisports sample site, you can create assetsets that contain articles and images. When coding your site pages, you code statements that create assetsets and then display the assets in them.

Searchstate

You identify which flex assets should be in an assetset by using the SEARCHSTATE method family in the templates for your flex assets. A *searchstate* is a set of search constraints that are applied to a list or set of flex assets. A constraint can be either a filter (restriction) based on the value of an attribute or based on another searchstate (called a nested searchstate).

A searchstate can search either the `_Mungo` tables in the database or the attribute indexes created by a search engine. This means that you can mix database and rich-text (full-text through an index) searches in the same query.

Because these tags search only the `_Mungo` table or attribute indexes for that flex asset type, using them to extract your flex assets is much more efficient than using the ASSET tags or the query asset.

Assetsets and Attribute Asset Types

WebCenter Sites cannot perform searches across attribute asset types. Because assetsets are created on the basis of attribute values, only assets that share the same attribute asset type can be included in the same assetset. This is an important point to consider when you design your flex families: data is separated for flex asset types that do not share a common attribute asset type. By creating such flex asset types you ensure that assets from different types cannot be included in a common assetset. And displaying assetsets is the mechanism for displaying flex assets on your delivery system.

For example, you can have two types of flex assets in the same flex family. While they use the same type of attributes, you can create assetsets that include assets of both types. Keep in mind, though, that a search across two types of flex assets creates a join between their `_Mungo` tables, which can deprecate performance.

In the avisports sample site there are two flex asset types: `article` and `image`. They share the same attribute asset type (`ContentAttribute`). This shared attribute asset type enables them to be included in the same assetset, even though they are two different flex asset types.

4.6 Search Engines and the Two Asset Models

Basic and flex asset models interact with search engines differently from each other because of the differences in their data structure. You'll use a different method in each model to index asset type fields for search.

- A basic asset type is defined by an asset descriptor file. Its primary storage table includes all of its properties as columns. To specify which fields of a basic asset type should be indexed, you must customize certain elements for the asset type. See Designing Basic Asset Types.

- Because fields for flex assets are flex attributes, which are assets, you decide which fields are indexed for rich-text search, attribute by attribute. Additionally, the WebCenter Sites application enables you to specify which attributes should be indexed with the **Search Engine** field on the attribute's New and Edit forms. You do not have to customize any elements to enable this feature.

4.7 Tags and the Two Asset Models

The assets you create and manage, ultimately you move them to the delivery system where the code in the elements extracts the assets from the database and display them to your site visitors. The WebCenter Sites applications offer various toolsets, custom tag sets, in both XML and JSP to help extract assets from the database.

The toolset you use to extract assets from the database in your templates depends on the kind of asset that you are working with.

- For assets with the basic asset model, use the `ASSET` method family.

- For the flex asset member in a flex family, use the `ASSETSET` and `SEARCHSTATE` method families. Note that you should not use the `ASSET.LOAD` tag for the flex asset member in a flex family (product, article, and image, for example). Using `ASSET.LOAD` tag for flex assets is extremely inefficient because it retrieves all of the information for that asset from all of its tables. The `SEARCHSTATE` methods queries only the `_Mungo` table for the asset type of the flex asset and the `MungoBlobs` table.

- For recommendation assets, use the `COMMERCECONTEXT` method family.

There are many more method families available with these products and an extensive set of custom tags from WebCenter Sites itself and several APIs.

For information about WebCenter Sites tags, see the *Oracle Fusion Middleware Tag Reference for Oracle WebCenter Sites Reference*.

4.8 Summary: Basic and Flex Asset Models

While the basic and flex models are quite different from each other, they are similar, too, in some ways.

The similarities and differences between the two asset models are as follows:

Where the Asset Models Intersect

Even though there are many differences in the way that the basic and flex asset models function, there are several points of intersection.

- No matter which asset model you are using, basic or flex, you use the template and page asset types that are delivered with the WebCenter Sites application.

- All asset types have a status code, which means that all assets, whether they are flex or basic, can be searched for with queries based on status.

- All asset types, whether they are flex or basic, have the following configuration or administrative traits in common:

  - They must be enabled by site.

  - They must have start menu items configured for them before anyone can create individual instances of those types.

  - Individual instances of them can be imported with the XMLPost utility.

Where the Asset Models Differ

Table 4-9 summarizes the major differences between the asset models.

Table 4-9   Major Differences Between the Asset Models

Feature	Basic Asset Model	Flex Asset Model
Number of database tables	One.	Several.
Adding fields to an asset type	Requires a schema change.	Does not require a schema change.
Links to other assets	Through associations and unnamed relationships.	Through flex family relationships.
Subtypes	Usually available through the **Subtype** item on the **Admin** tab. See Subtype.	Through flex definitions and flex parent definitions.
Search engine indexing	Must customize certain elements for the asset type.	Use the **Search Engine** field in the flex attribute form.
Main tag families	`ASSET`, `SITEPLAN`, and `RENDER`.	`ASSETSET`, `SEARCHSTATE`, and `RENDER`.
Publishing methods	Export to Disk. Mirror to Server.	Export to Server is possible, but is atypical for the flex model. Mirror to Server.

5

Designing Basic Asset Types

The AssetMaker utility is available for you to create basic asset types. You also need to create an asset descriptor file and asset table, add subtypes, set association fields, etc., before you migrate the asset types to the management/delivery system for content and marketing teams.

Topics:

- About the AssetMaker Utility

- Before You Begin Creating Basic Asset Types

- Creating Basic Asset Types

- Deleting Basic Asset Types

5.1 About the AssetMaker Utility

To create basic asset types, the AssetMaker utility carries out several operations in AssetType, SystemInfo, Category, Catalog, and SystemSQL tables using a descriptor file which you'll create.

Before the AssetMaker utility can begin its work, it needs a descriptor file that defines properties for the new asset type. The term *property* means both a column in a database table and a field in a WebCenter Sites entry form. So, first define the basic asset type inWebCenter Sites an XML file–*asset descriptor file* using AssetMaker XML tags.

Your next step is to upload the file to WebCenter Sites and create two items using AssetMaker: a database table for the new asset type, and the WebCenter Sites elements which generate the forms that you and others use when working with assets of the new type (creating, editing, copying, and so on).

This section includes the following topics:

- How AssetMaker Works

- Asset Descriptor Files

- Columns in the Asset Type's Database Table

- Elements and SQL Statements for the Asset Type

5.1.1 How AssetMaker Works

Creating a new basic asset type using AssetMaker involves the following steps:

1. Code the asset descriptor file.

This chapter describes asset descriptor files and coding them. For information about AssetMaker tags, see the *Oracle Fusion Middleware Tag Reference for Oracle WebCenter Sites Reference*.

2. Upload the file.

   AssetMaker creates a row in the `AssetType` table and copies the asset descriptor file to that row.

3. Create the table.

   When you click the **Create Asset Table** button, AssetMaker does the following:

   - Parses the asset descriptor file.

   - Creates the primary storage table for assets of that type. The name of the table matches the name of the asset type identified in the asset descriptor file. The data type of each column is defined by statements in the file as well.

     In addition to the columns defined in the asset descriptor file, AssetMaker creates default columns that WebCenter Sites needs to function correctly.

   - Adds a row for the new table to the `SystemInfo` table.

     All asset tables are object tables so the value in the `systable` column is set to `obj`.

     All asset tables have URL columns. So, the value in the `defdir` column is set to the value that you specified either in the asset descriptor file or in the `DefDir` field in the Create Asset Table form when you create the asset type.

   - If you have enabled the **Add General Category** option, Asset Maker adds one row to the `Category` table for the new asset type and names that category `General`.

4. Register the elements.

   AssetMaker does the following:

   - Creates a subdirectory in the `ElementCatalog` table under `OpenMarket/Xcelerate/AssetType` directory for the new asset type.

   - Copies elements from the `AssetStubCatalog` table to the new subdirectory in the `ElementCatalog` table. These elements render WebCenter Sites forms for working with assets of this type and provide the processing logic for the WebCenter Sites functions.

   - Creates SQL statements that implement searches on individual fields in the search forms. These statements are placed in the `SystemSQL` table.

When you create, edit, inspect, and so on an asset of this type, AssetMaker parses the asset descriptor file located in the `AssetType` table, and passes its values to WebCenter Sites so that the forms are specific to the asset type. Statements in the asset descriptor file determine the input types of the fields, specify field length restrictions, and determine whether the field is displayed on search and search results forms.

Note that after you create an asset type, you must enable the asset type on the sites that will use it, create start menu shortcuts, and so on.

The following figure shows a flow chart that summarizes how AssetMaker works, and which database tables are involved when a basic asset type is created.

Figure 5-1   How AssetMaker Works

User Action	AssetMaker's Response	Database Tables

1. Code asset descriptor file

Asset descriptor file

2. Upload asset descriptor file

AssetMaker enters the asset descriptor file into a row of the ..

AssetType table

3. Create the asset storage table

Using the asset descriptor file, AssetMaker creates the...

Primary storage table for the asset type

AssetMaker adds a row for the new storage table to the...

SystemInfo table

If you checked the **Add 'General' Category** checkbox

AssetMaker adds a category row named "General" to the...

Category table

copies elements from

AssetStub Catalog Table

to

Subdirectory of Element Catalog table

4. Register the elements

AssetMaker creates a subdirectory in the ElementCatalog table, and ...

creates SQL statements and copies them to the ...

SystemSQL table

for use in

Search forms

5.1.2 Asset Descriptor Files

Using the AssetMaker XML tags, you code asset descriptor files that define the asset types you design for your systems.

This section includes the following topics:

- About the Asset Descriptor File

- About Format and Syntax

- About the AssetMaker Tags

5.1.2.1 About the Asset Descriptor File

An asset descriptor file is a valid XML document in which developers define a basic asset type using AssetMaker tags. An asset descriptor file does the following:

- Describes the asset type in terms of data structure. It specifies the name of the database table, the names of the columns, the columns data types, and the sizes of the fields on the WebCenter Sites forms.

- Formats the HTML forms that are displayed by WebCenter Sites when users work with assets of the given type. Formatting an HTML form means naming the fields on the form, displaying the fields in required format (for example, check box, radio button, or drop-down list), accounting for field specifications (such as the number of characters that can be entered in to a text field), and so on.

AssetMaker uses the asset descriptor file to create a database table for the new asset type. When content providers work with assets of the given type (create, edit, and so on), AssetMaker parses the asset descriptor file, using the data in the file to customize the forms that WebCenter Sites displays.

> **Note:**
>
> For reference, sample AssetMaker descriptor code is provided on the WebCenter Sites installation medium, in the **Samples** folder. The same folder contains the readme.txt file that describes the sample descriptor files.

5.1.2.2 About Format and Syntax

The basic format for every asset descriptor file is shown below. To the right of each AssetMaker tag is a brief description of the tag.

```
<?xml version="1.0" ?>
<ASSET ...> Names the asset type (storage table)
<PROPERTIES> Starts the properties specification section
  <PROPERTY ...> Specifies column and field name for the property
    <STORAGE .../> Specifies data type for the column
    <INPUTFORM .../> Specifies field format on New, Edit, Inspect forms
    <SEARCHFORM .../> Specifies field format on Advanced Search form
    <SEARCHRESULTS .../> Specifies which fields are shown in search results
  </PROPERTY>
  <PROPERTY ...>
    <STORAGE .../>
    <INPUTFORM .../>
    <SEARCHFORM .../>
    <SEARCHRESULTS .../>
  </PROPERTY>
  <PROPERTY ...>
...
</PROPERTIES> Ends the properties specification section
</ASSET> Ends the asset descriptor file
```

Shown next is the syntax of an asset descriptor file, indicating some parameters that an AssetMaker tag can take:

```
<?xml version="1.0" ?>
<ASSET NAME="assetTypeName" DESCRIPTION=""assetTypeName" ...>
<PROPERTIES>
  <PROPERTY NAME="fieldName1" DESCRIPTION="fieldName1"/>
    <STORAGE TYPE="VARCHAR" LENGTH="36"/>
    <INPUTFORM TYPE="TEXT" DESCRIPTION="fieldName1".../>
    <SEARCHFORM TYPE="TEXT" DESCRIPTION="fieldName1".../>
    <SEARCHRESULTS INCLUDE="TRUE"/>
  </PROPERTY>
  <PROPERTY NAME="fieldName2" DESCRIPTION="fieldName2"/>
    <STORAGE TYPE="INTEGER" LENGTH="4"/>
    <INPUTFORM TYPE="TEXT" DESCRIPTION="fieldName2".../>
    <SEARCHFORM TYPE="TEXT" DESCRIPTION="fieldName2".../>
   <SEARCHRESULTS INCLUDE="TRUE"/>
  </PROPERTY>
...
</PROPERTIES>
</ASSET>
```

For information about WebCenter Sites tags and their parameters, along with sample code, see the *Oracle Fusion Middleware Tag Reference for Oracle WebCenter Sites Reference*.

5.1.2.3 About the AssetMaker Tags

- An asset descriptor file begins with the standard XML version tag:

  ```
  <?xml version="1.0"?>
  ```

- The ASSET tag, which follows the XML version tag, names the asset type and therefore its storage table in the WebCenter Sites database. The ASSET tag also sets some behavior and display attributes of assets of that type; for example, the ASSET tag determines what graphical notation designates that a field is required, and whether an asset can be previewed.

  The opening tag <ASSET> is always the first line of code and the closing tag <\ASSET> is always the last line of code in the asset descriptor file. Note that there is only one ASSET tag pair in each asset descriptor file because only one asset type per asset descriptor file can be created.

- The PROPERTIES tag marks the section of the file that holds the property descriptions. The opening tag <PROPERTIES> is always the second statement in the asset descriptor file. There is only one PROPERTIES tag pair in each asset descriptor file.

  > **Note:**
  >
  > The PROPERTIES tag is required in every asset descriptor file, even if no PROPERTY tags are needed.

- The PROPERTY tags, nested within the PROPERTIES tag pair, specify the columns and fields for assets of this type. Each PROPERTY tag specifies the database name of the column that will hold the value(s) users will enter for this property, and the column's display name (that is, its field name) as it will appear on the form that will be rendered for users who are to work with assets of this type.

- Nested inside each pair of `PROPERTY` tags are the following tags:

    - `STORAGE`: Specifies the data type of the column that is being established by this property. Note that the data type in the `STORAGE` tag must map to one of the data types that is defined by the properties on the database entry of the `futuretense.ini` file.

    - `INPUTFORM`: Specifies the name and format of the field on the New, Edit, and Inspect forms. For example, whether the field is a drop-down list or a check box or a text field. The field's input type must be compatible with the data type of the database column, as specified by the `STORAGE` statement.

    - `SEARCHFORM`: Specifies the format of the field (property) when it displays on the Advanced Search form. Omitting the `SEARCHFORM` statement from the `PROPERTY` section prevents the field being defined from appearing on the Advanced Search form.

        Setting the `TYPE` parameter values to `Table` or `Date` displays a drop-down list on the Advanced Search form for the asset type, but not on the SimpleSearch form.

    - `SEARCHRESULTS`: Specifies which fields are displayed in the search results form after a search is run. The field value is also displayed if the `INCLUDE` parameter is set to `true`. This tag is optional.

        When modifying a standard field, do not set `SEARCHRESULTS` to `true` for name or description.

For information about AssetMaker tags and their parameters, see the *Oracle Fusion Middleware Tag Reference for Oracle WebCenter Sites Reference*. That section also provides information about dependencies and restrictions among the parameters `STORAGE TYPE`, `INPUTFORM TYPE`, and `SEARCHFORM TYPE`.

5.1.3 Columns in the Asset Type's Database Table

When AssetMaker creates the database table for a new asset type, it creates columns for all the properties defined by the `PROPERTY` tags in the asset descriptor file, and it creates default columns that are required by WebCenter Sites for its basic functionality. For a list of the default columns in each asset type's table, see Default Columns in the Basic Asset Type Database Table.

This section includes the following topics:

- The Source Column: A Special Case

- Storage Types for the Columns

- Input Types for the Fields

- Data Types for Standard Asset Fields

5.1.3.1 The Source Column: A Special Case

All of the asset type tables can also have a `source` column. WebCenter Sites provides a `Source` table and a Source form in the **Admin** node on the **General Admin** tree. that you use to add the rows to the `Source` table. You can use this feature to identify where an asset originated. However, unlike the columns listed in the preceding table, the `source` column is not automatically created when AssetMaker creates the asset type table. To add the source column to your table and have it displayed on your asset

forms, you must include a PROPERTY description for it in the asset descriptor file. See Example 5-2.

5.1.3.2 Storage Types for the Columns

The STORAGE TYPE parameter specifies the data type of a column. The data types are defined by the WebCenter Sites database properties located in the wcs_properties.json file.

The following table presents the possible data types for your asset type's table columns.

Table 5-1   STORAGE TYPE Parameter

Type (generic ODBC/JDBC data type)	Property
CHAR	cc.char
VARCHAR	cc.varchar
SMALLINT	cc.smallint
INTEGER	cc.integer
BIGINT	cc.bigint
DOUBLE	cc.double
TIMESTAMP	cc.datetime
BINARY	cc.blob
LONGVARCHAR	cc.bigtext

5.1.3.3 Input Types for the Fields

The INPUT TYPE parameter specifies how data can be entered in a field when it is displayed in the WebCenter Sites forms. The following table lists all the input types. Note that the input type for a field must be compatible with the data type of its column.

Table 5-2   INPUT TYPE Parameter

Input TYPE	Description
TEXT	A single line of text.
	Corresponds to the HTML input type named TEXT.

Table 5-2   (Cont.) INPUT TYPE Parameter

Input TYPE	Description
TEXTAREA	A text box, with scroll bars, that accepts multiple lines of text.
	Corresponds to the HTML input type named TEXTAREA.
	To accommodate large amounts of text in the field, create a text box that displays the contents of a URL column. To do so, you must specify a string for PROPERTY NAME that begins with url and set the STORAGE TYPE to VARCHAR.
	When a user clicks **Save**, the text entered into this kind of field is stored in the file directory specified as the default storage directory for this asset type. You can specify the default storage directory (defdir) in either the asset descriptor file, or in the AssetMaker form when you create the asset type.
	Note:
	• You can specify an unlimited size for a url field that is edited using a TEXTAREA field by not specifying a value for the MAXLENGTH parameter.
	• Do not use the following suffixes with a string for PROPERTY NAME that begins with the letters url: _type, _size, _folder, and _file.
UPLOAD	A field that takes a file name (a URL) and presents a **Browse** button so that you can either enter the path to and name of a file or browse to it and select it.
	When you specify that a field is an upload field, set a string for PROPERTY NAME that begins with url and set STORAGE TYPE (the property's data type) to VARCHAR.
	You can also use the BLOB storage type for an upload field. In this case, the PROPERTY NAME string does not have to begin with url.
	When the user clicks **Save**, WebCenter Sites uploads the selected file and stores it in the file directory specified as the default storage directory for this asset type. You can specify the default storage directory (defdir) in either the asset descriptor file, or in the AssetMaker form when you upload the file.
	Note:
	• The size of a file that is selected in an upload field cannot exceed 30 megabytes.
	• If you specify a string for PROPERTY NAME that begins with the letters url, do not use the following suffixes: _type, _size, _folder, and _file.
SELECT	A field that presents a drop-down list of options that can be selected.
	You can either specify the options that are presented in the list or you can specify a query so that the options are selected from the database (or an external table) and presented dynamically.
	Corresponds to the HTML input type SELECT.

Table 5-2   (Cont.) INPUT TYPE Parameter

Input TYPE	Description
CHECKBOX	A check box field. You can specify the names of the check box options or you can specify a query so that the names are selected from the database (or an external table) and presented dynamically. This input type allows the user to select multiple options. Corresponds to the HTML input type CHECKBOX.
RADIO	A radio button control. You can either specify the names of the radio options or you can specify a query so that the names are selected from the database (or an external table) and presented dynamically. This input type allows the user to select only one option. Corresponds to the HTML input type RADIO.
CKEDITOR	A field whose contents you edit by using the CKEditor text editor. When you specify that a field is a CKEditor field, it is recommended that you make it a URL field. That is, set a string for PROPERTY NAME with the url prefix and set STORAGE TYPE (the property's data type) to VARCHAR. If you specify a string form PROPERTY NAME that begins with the url prefix, do not use the following suffixes: _type, _size, _folder, and _file.

ELEMENT	Calls an element that you create to display a field on the ContentForm, ContentDetails, or SearchForm forms. The custom element must be found at one of the following locations:
	
	• For a field on the ContentForm form:
	`OpenMarket/Xcelerate/AssetType/myAssetType/ContentForm/fieldname`
	
	• For a field on the ContentDetails form:
	`OpenMarket/Xcelerate/AssetType/myAssetType/ContentDetails/fieldname`
	
	• For a field on the SearchForm form:
	`OpenMarket/Xcelerate/AssetType/myAssetType/SearchForm/fieldname`
	
	Where *myAssetType* is the asset type that you are creating the custom field for, and fieldname is the name of the custom field.
	
	An ELEMENT field can have any storage type, including BLOB.

5.1.3.4 Data Types for Standard Asset Fields

You can customize the appearance of the WebCenter Sites standard asset fields. All other changes are conditional on the type of field, as described below:

• All standard fields. You can change their display names.

• A standard field that is not a system field. You must not change its data type, with one exception: You can change only the length of the VARCHAR data type.

• System fields. You must not change the data type (including the length of a VARCHAR type of field).

The following table lists the data types of standard fields (and indicates whether they are also system fields).

Table 5-3   Data Types for Standard Asset Fields

Standard Field	System Field	Data Type
ID	Yes	NOT NULL NUMBER(38)
NAME	N/A	NOT NULL VARCHAR(64)
DESCRIPTION	N/A	VARCHAR(128)
TEMPLATE	Yes	VARCHAR(64)
SUBTYPE	N/A	VARCHAR(24)
FILENAME	N/A	VARCHAR(64)
PATH	N/A	VARCHAR(255)
STATUS	Yes	NOT NULL VARCHAR(2)
EXTERNALDOCTYPE	Yes	VARCHAR(64)
URLEXTERNALDOCXML	Yes	VARCHAR(255)
URLEXTERNALDOC	Yes	VARCHAR2(255)
CREATEDBY	Yes	NOT NULL VARCHAR(64)
UPDATEDBY	Yes	NOT NULL VARCHAR(64)
CREATEDDATE	Yes	NOT NULL DATE
UPDATEDDATE	Yes	NOT NULL DATE
STARTDATE	N/A	DATE
ENDDATE	N/A	DATE

5.1.4 Elements and SQL Statements for the Asset Type

After you upload an asset descriptor file, you register the elements. When you register elements, AssetMaker copies elements in the `AssetStubElementCatalog` table to a directory in the `ElementCatalog` table for this asset type. Additionally, AssetMaker copies several SQL statements that implement the WebCenter Sites searches on the Simple Search and the Advanced Search forms for assets of this type.

If necessary, you can customize the SQL statements, the asset type-specific elements, or, in some cases, the elements in the `AssetStubElementCatalog` table. See Customizing the Asset Type Elements (Optional) .

> **Note:**
>
> **Under no circumstances** should you modify any of the other WebCenter Sites elements.

Topics:

- The Elements
- The SQL Statements

5.1.4.1 The Elements

AssetMaker places the elements for your new asset type to the `ElementCatalog` table according to the following naming convention:

```
OpenMarket/Xcelerate/AssetType/YourNewAssetType
```

For example, the elements for the sample asset type ImageFile are located here:

```
OpenMarket/Xcelerate/AssetType/ImageFile
```

The following table lists the elements that AssetMaker copies for each asset type.

Table 5-4   AssetMaker Elements

Element	Description
ContentForm	Renders the New and Edit forms for assets of this type.
	When the function is invoked, AssetMaker uses the INPUTFORM statements in the asset descriptor file to format these forms.
ContentDetails	Formats the Inspect form for assets of this type.
	When the function is invoked, AssetMaker uses the INPUTFORM statements in the asset descriptor file to customize these forms.
SimpleSearch	Renders the Simple Search form for assets of this type.
	When the function is invoked, AssetMaker uses the SEARCHFORM statements in the asset descriptor file to format these forms.
SearchForm	Formats the Advanced Search form for assets of this type.
	When the function is invoked, AssetMaker uses the SEARCHFORM statements in the asset descriptor file to format these forms.
AppendSelectDetails	Builds the SQL queries on the individual fields in the Advanced Search form.
	When the Advanced Search form is rendered, AssetMaker uses the SEARCHFORM statements in the asset descriptor file to customize the form.
AppendSelectDetailsSE	Builds the SQL queries on the individual fields in the Advanced Search form when your system is using an external search engine.
	When this function is invoked, AssetMaker uses the SEARCHFORM statements in the asset descriptor file to create the SQL queries.
IndexAdd	The IndexAdd and IndexReplace elements establish which fields (columns) are indexed by the search engine when you are using a search engine. By default, only the standard fields are indexed. To index other fields, you must customize these forms. See Customizing the Asset Type Elements (Optional).
IndexReplace	See the description of IndexAdd.

Table 5-4   (Cont.) AssetMaker Elements

Element	Description
Tile	Formats the Search Results page, a page that lists the assets that meet the search criteria, for assets of this type.
	When the page is rendered, AssetMaker uses the SEARCHRESULTS statements in the asset descriptor file to display the results.
LoadTree	Determines how assets of this type appear when they are displayed on any tab in the tree other than the **Site Navigation** tab.
LoadSiteTree	Determines how assets of this type appear when they are displayed on the **Site Navigation** node in the **Site** tree.
PreUpdate	Is called before a function that writes to the database is completed. In other words, before an asset is saved and during the create, edit, delete, or XMLPost functions, this element is called.
	This element takes no input from the asset descriptor file. However, you can customize it directly.
PostUpdate	Is called after a function that writes to the database is completed. In other words, after an asset is created, edited, deleted, or imported with XMLPost, this element is called.
	You can customize this element.

5.1.4.1.1 About PreUpdate and PostUpdate Elements

Actions or procedures that can be performed on assets are called functions. For example, New, Edit, and Delete are all functions that can be invoked by users of the Admin and Oracle WebCenter Sites: Contributor interfaces to create, edit, and delete assets. Such functions also call the PreUpdate and PostUpdate elements.

PreUpdate and PostUpdate elements are used to contain logic that initiates various operations when the elements are called. The PreUpdate element is called when a function is invoked; the PostUpdate element is called after WebCenter Sites writes asset information to the database. Each element contains a variable whose name and value specify conditions that call the element. To call the element from the Admin interface or from Form Mode of the Contributor interface or the XMLPost utility, name the variable as updatetype. To call the element from Web Mode of the Contributor interface, name the variable as servicesUpdateType. For example, your content managers are working with the Admin interface and their system is configured to import batches of articles from a wire service. You can have the PreUpdate element set the value for the **Source** field to wirefeed and the value for the **Byline** field to API just before import occurs if you code these operations in the element and set updatetype=remotepost as the condition under which the element will be called.

> **Note:**
>
> PreUpdate and PostUpdate elements are always called when a WebCenter Sites user invokes the New, Edit, or Delete function in the Admin or Contributor interface, or when assets are imported through XMLPost. Whether operations are performed using the PreUpdate and PostUpdate elements depends on how the elements are coded. By default, they are designed for no action.

PreUpdate and PostUpdate elements are accessible from Explorer, in the following path:

ElementCatalog\OpenMarket\Xcelerate\AssetType

Note:

The PreUpdate element is called twice if a user saves a new or edited asset in the Admin interface:

The first call occurs before the New or Edit form is rendered.The second call occurs after the user clicks **Save** in the New or Edit form, but WebCenter Sites has not yet written asset information to the database.

The condition above provides the opportunity to perform operations using PreUpdate at one or more points once the New or Edit function is invoked: before the New or Edit form is rendered, before asset information is written to the database, or both, depending on the value of the element's updatetype variable.

The following table defines the values of the updatetype variable:

Table 5-5   Values of the UpdateType Variable

updatetype =	Description
setformdefaults	When a user invokes the **New** function in the Admin interface or in the Form Mode of the Contributor interface. • The PreUpdate element is called before the New form is rendered. • For PostUpdate, setformdefaults is not a legal value for the updatetype variable.
create	When a user saves a new asset in the Admin interface or in the Form Mode of the Contributor interface: • The PreUpdate element is called before the new asset is written to the database. • The PostUpdate element is called after the new asset is written to the database.
editfront	When a user invokes the **Edit** function in the Admin interface or in the Form Mode of the Contributor interface: • The PreUpdate element is called before the Edit form is rendered. • For PostUpdate, editfront is not a legal value for the UpdateType variable.
edit	When a user saves an edited asset in the Admin interface or in the Form Mode of the Contributor interface: • The PreUpdate element is called before the edits are written to the database. • The PostUpdate element is called after the edits are written to the database.

Table 5-5   (Cont.) Values of the UpdateType Variable

updatetype =	Description
delete	When a user deletes an asset from the Admin interface: • The PreUpdate element is called before WebCenter Sites deletes the asset. • The PostUpdate is performed after WebCenter Sites deletes the asset.
remotepost	When a user invokes the XMLPost function to import an asset: • The PreUpdate element is called before the asset is imported. • The PostUpdate element is called after the asset is imported.
InSite	When saved from Web Mode, a variable called servicesUpdateType will have create, edit, or delete depending on the user operation. Table 5-6 defines the values of the servicesUpdateType variable.

The following table defines the values of the servicesUpdateType variable.

Table 5-6   Values of the servicesUpdateType Variable (Contributor interface)

servicesUpdateType =	Description
create	When a user saves a new asset in the Contributor interface: • The PreUpdate element is called before the new asset is written to the database. • The PostUpdate element is called after the new asset is written to the database.
edit	When a user saves an edited asset in the Contributor interface: • The PreUpdate element is called before the edits are written to the database. • The PostUpdate element is called after the edits are written to the database.
delete	When a user deletes an asset in the Contributor interface: • The PreUpdate element is called before WebCenter Sites deletes the asset. • The PostUpdate element is called after WebCenter Sites deletes the asset.

5.1.4.2 The SQL Statements

AssetMaker places the SQL statements in the SystemSQL table according to the following naming convention:

OpenMarket/Xcelerate/AssetType/YourNewAssetType

For example, the elements for the sample asset type ImageFile are located here:

OpenMarket/Xcelerate/ImageFile

The following table lists the SQL elements that AssetMaker creates:

Table 5-7   SQL Elements

Statement	Description
SelectSummary	A SQL statement that defines the query used in the Simple Search and Advanced Search form for assets of this type.
SelectSummarySE	Not used.

5.2 Before You Begin Creating Basic Asset Types

To be able to create an efficient asset type design, you need to consider many points such as the number of fields, data types, asset associations. Also set up your development system with a CM site and a user Id with appropriate rights.

- Planning the Asset Type Design

- Setting Up Your Development System

5.2.1 Planning the Asset Type Design

Be sure to design your asset types on paper before you start coding an asset descriptor file. Consider the following kinds of details:

- What fields do you need?

  In general, try to minimize the number of fields that you use by organizing the information into useful units. When determining those units, consider both the information you plan to display on your online site and the data-entry needs of the content providers who will enter that data.

- What is the appropriate data type for each field?

- For fields with options, how will you supply the options?

  With a static list coded in the asset descriptor file or with a lookup table that holds the valid options?

- Which WebCenter Sites features will you use to organize or categorize assets of this type?

  For example, source, category, and asset associations. For each one, determine its name and plan how it will be used both on the management system and in the design of your online site.

- Does the implementation of your site design require assets of this type to use a different default template based on the publishing target that they are published to?

  If so, you will have to use the **Subtype** feature. Determine the names of the subtypes that you will need for assets of this type.

5.2.2 Setting Up Your Development System

Also before you begin, be sure to set up your development system. For information about any of these preliminary steps, see *Oracle Fusion Middleware Administering Oracle WebCenter Sites*.

- Create the appropriate sites.

- Create a user name for yourself that has administrator rights and enable that user name on all of the sites on your development system. Be sure that the `TableEditor` ACL is assigned to your user name or you will be unable to create asset types.

> **Note:**
>
> Without administrator rights, you do not have access to the **Admin** node in the **General Admin** tree, which means that you cannot perform any of the procedures in this chapter. For the sake of convenience, assign the `Designer` and `GeneralAdmin` roles to your user name. That way you will have access to all the tabs and all of the existing Start Menu shortcuts for the assets in the sample site.

5.3 Creating Basic Asset Types

You spend less time creating a simple asset type that needs just one descriptor file. For some asset types you can modify the code in the elements that AssetMaker creates, or add a database table to hold information for the dropdown lists.

Creating basic asset types includes these tasks:

- Coding the Asset Descriptor File.

- Uploading the Asset Descriptor File to WebCenter Sites using AssetMaker in the **Admin** tab.

- Creating the Asset Table and register the asset type elements by copying the asset type elements from the `AssetStubElementCatalog` table to the appropriate directory in the `ElementCatalog` table.

- Configuring the Asset Type.

- Enabling the Asset Type on Your Site and create a start menu shortcut so that you can work with the asset type.

- Fine-Tuning the Asset Descriptor File (if necessary) and re-register the asset type elements.

- Customizing the Asset Type Elements (Optional) .

- Adding Subtypes (Optional) for the new asset type.

- Configuring Association Fields (Optional) for the new asset type.

- Configuring Categories (Optional) for the new asset type.

- Adding Mimetypes (Conditional) for the new asset type.

- Editing Search Elements to Enable Indexed Search (Optional) if you are using a search engine rather than the WebCenter Sites database search utility to perform the logic behind the search forms and you want to use it on your new asset type.

- Creating and Assigning Asset Type Icons (Contributor Interface Only) that will represent the asset type in the Contributor interface's navigation trees (**Site Tree**, **Content Tree**, and **My Work** tree).

- Coding Templates for the Asset Type, Coding Templates for the Asset Type. See also Coding Elements for Templates and CSElements.

- Moving the Asset Types to Other Systems (management and delivery) This allows your administrator to complete the final steps in creating the asset type, including setting up workflow and creating start menu items.

5.3.1 Coding the Asset Descriptor File

As described in Asset Descriptor Files, this is the basic format of an asset descriptor file:

```xml
<?xml version="1.0" ?>
<ASSET NAME="assetName"...>
<PROPERTIES>
  <PROPERTY.../>
    <STORAGE.../>
    <INPUTFORM.../>
    <SEARCHFORM.../>
    <SEARCHRESULTS.../>
  </PROPERTY>
  <PROPERTY... />
    <STORAGE.../>
    <INPUTFORM.../>
    <SEARCHFORM.../>
    <SEARCHRESULTS.../>
  </PROPERTY>
</PROPERTIES>
</ASSET>
```

To code your asset descriptor files, see the *Oracle Fusion Middleware Tag Reference for Oracle WebCenter Sites Reference* and use the tags described in this guide to code the file. Use the native XML editor in Explorer or any other XML editor to code the file.

Note that you can customize the appearance of standard asset fields by including them in your asset descriptor file. Changing a field's storage type is conditional. For example, a system field's storage type must not be changed. For the list of standard fields, their storage types, and allowed changes to storage type, see Data Types for Standard Asset Fields.

This section offers a sample asset descriptor file and several examples about coding specific kinds of properties.

This section includes the following examples:

- Example 5-1

- Example 5-2

- Example 5-3

- Example 5-4

- Example 5-5

- Example 5-6

- Example 5-7

- Example 5-8

- [Example 5-9](#)

- [Example 5-10](#)

Example 5-1   Sample Asset Descriptor File: ImageFile.xml

An example of an Asset Descriptor File, `ImageFile.xml`, follows:

```
<!-- this is the description of an asset -->
<ASSET NAME="ImageFile" DESCRIPTION="ImageFile"
 MARKERIMAGE="/Xcelerate/data/help16.gif" PROCESSOR="4.0"
 DEFDIR="c:\FutureTense\Storage\ImageFile">

<PROPERTIES>

  <PROPERTY NAME="source" DESCRIPTION="Source">
    <STORAGE TYPE="VARCHAR" LENGTH="24"/>
    <INPUTFORM DESCRIPTION="Source" TYPE="SELECT" TABLENAME="Source"
     OPTIONDESCKEY="description" OPTIONVALUEKEY="source" SOURCETYPE="TABLE"/>
    <SEARCHFORM DESCRIPTION="Source" TYPE="SELECT" TABLENAME="Source"
     OPTIONDESCKEY="description" OPTIONVALUEKEY="source" SOURCETYPE="TABLE"/>
  </PROPERTY>

  <PROPERTY NAME="urlpicture" DESCRIPTION="Image File">
    <STORAGE TYPE="VARCHAR" LENGTH="255"/>
    <INPUTFORM TYPE="UPLOAD" WIDTH="36" REQUIRED="NO" LINKTEXT="Image"/>
  </PROPERTY>

  <PROPERTY NAME="urlthumbnail" DESCRIPTION="Thumbnail File">
    <STORAGE TYPE="VARCHAR" LENGTH="255"/>
    <INPUTFORM TYPE="UPLOAD" WIDTH="36" REQUIRED="NO" LINKTEXT="Image"/>
  </PROPERTY>

  <PROPERTY NAME="mimetype" DESCRIPTION="Mimetype">
    <STORAGE TYPE="VARCHAR" LENGTH="36"/>
    <INPUTFORM TYPE="SELECT" SOURCETYPE="TABLE" TABLENAME="MimeType"
     OPTIONDESCKEY="description" OPTIONVALUEKEY="mimetype"
     SQL="SELECT mimetype, description
     FROM MimeType
     WHERE keyword = 'image'
     AND isdefault = 'y'"
     INSTRUCTION="Add more options to mimetype table with isdefault=y
     and keyword=image"/>
    <SEARCHFORM DESCRIPTION="MimeType" TYPE="SELECT" SOURCETYPE="TABLE"
     TABLENAME="MimeType" OPTIONDESCKEY="description" OPTIONVALUEKEY="mimetype"
     SQL="SELECT mimetype, description
     FROM MimeType
     WHERE keyword = 'image'
     AND isdefault = 'y'"/>
  </PROPERTY>

  <PROPERTY NAME="width" DESCRIPTION="Width">
    <STORAGE TYPE="INTEGER" LENGTH="4"/>
    <INPUTFORM TYPE="TEXT" WIDTH="4" MAXLENGTH="4" REQUIRED="NO" DEFAULT=""/>
    <SEARCHFORM DESCRIPTION="Width is" TYPE="TEXT" WIDTH="4"
     MAXLENGTH="4" VERB="="/>
  </PROPERTY>

  <PROPERTY NAME="height" DESCRIPTION="Height">
    <STORAGE TYPE="INTEGER" LENGTH="4"/>
    <INPUTFORM TYPE="TEXT" WIDTH="4" MAXLENGTH="4" REQUIRED="NO" DEFAULT=""/>
    <SEARCHFORM DESCRIPTION="Height is" TYPE="TEXT" WIDTH="4"
```

```
          MAXLENGTH="4" VERB="="/>
      </PROPERTY>

      <PROPERTY NAME="align" DESCRIPTION="Alignment">
        <STORAGE TYPE="VARCHAR" LENGTH="8"/>
        <INPUTFORM TYPE="SELECT" SOURCETYPE="STRING"
         OPTIONVALUES="Left,Center,Right" OPTIONDESCRIPTIONS="Left,Center,Right"/>
        <SEARCHFORM DESCRIPTION="Alignment" TYPE="SELECT" SOURCETYPE="STRING"
         OPTIONVALUES="Left,Center,Right" OPTIONDESCRIPTIONS="Left,Center,Right"/>
      </PROPERTY>

      <PROPERTY NAME="artist" DESCRIPTION="Artist">
        <STORAGE TYPE="VARCHAR" LENGTH="64"/>
        <INPUTFORM TYPE="TEXT" WIDTH="36" MAXLENGTH="36" REQUIRED="NO" DEFAULT=""/>
        <SEARCHFORM DESCRIPTION="Artist contains" TYPE="TEXT"
         WIDTH="36" MAXLENGTH="64"/>
      </PROPERTY>

      <PROPERTY NAME="alttext" DESCRIPTION="Alt Text">
        <STORAGE TYPE="VARCHAR" LENGTH="255"/>
        <INPUTFORM TYPE="TEXT" WIDTH="48" MAXLENGTH="255" REQUIRED="NO" DEFAULT=""/>
        <SEARCHFORM DESCRIPTION="Alt Text contains" TYPE="TEXT"
         WIDTH="48" MAXLENGTH="255"/>
      </PROPERTY>

      <PROPERTY NAME="keywords" DESCRIPTION="Keywords">
        <STORAGE TYPE="VARCHAR" LENGTH="128"/>
        <INPUTFORM TYPE="TEXT" WIDTH="48" MAXLENGTH="128" REQUIRED="NO" DEFAULT=""/>
        <SEARCHFORM DESCRIPTION="Keywords contain" TYPE="TEXT"
         WIDTH="48" MAXLENGTH="128"/>
      </PROPERTY>

      <PROPERTY NAME="imagedate" DESCRIPTION="Image date">
        <STORAGE TYPE="TIMESTAMP" LENGTH="8"/>
        <INPUTFORM TYPE="ELEMENT" WIDTH="24" MAXLENGTH="48" REQUIRED="NO"
         DEFAULT="" INSTRUCTION="Format: yyyy-mm-dd hh:mm"/>
        <SEARCHFORM DESCRIPTION="Image date" TYPE="ELEMENT"
         WIDTH="48" MAXLENGTH="128"/>
      </PROPERTY>

  </PROPERTIES>
</ASSET>
```

Example 5-2   Adding the Source Column and Field

The source column is not created by default even though WebCenter Sites has a **Source** feature on the **Admin** node in the **General Admin** tree. To use the Source feature on your new asset types, you must include a property statement for the source column and field.

Note the following:

- STORAGE TYPE must be set to VARCHAR, and LENGTH must be set to 24.

- INPUTFORM SOURCETYPE must be set to TABLE, and TABLENAME must be set to Source.

For example:

```
<PROPERTY NAME="source" DESCRIPTION="Source">
  <STORAGE TYPE="VARCHAR" LENGTH="24"/>
  <INPUTFORM TYPE="SELECT" TABLENAME="Source"
```

```
    OPTIONDESCKEY="description"
    OPTIONVALUEKEY="source" SOURCETYPE="TABLE"/>
  <SEARCHFORM DESCRIPTION="Source" TYPE="SELECT"
    TABLENAME="Source" OPTIONDESCKEY="description"
    OPTIONVALUEKEY="source" SOURCETYPE="TABLE"/>
</PROPERTY>
```

Example 5-3   Creating a Standard Upload Field

To create an upload field with a **Browse** button, code the PROPERTY statement as follows:

1. The string set for PROPERTY NAME must begin with the letters url.

2. The value for STORAGE TYPE must be set to VARCHAR.

3. The value for INPUT TYPE must be set to UPLOAD.

Here is a code snippet of an upload field from the ImageFile asset descriptor file:

```
<PROPERTY NAME="urlpicture" DESCRIPTION="Image File">
  <STORAGE TYPE="VARCHAR" LENGTH="255"/>
  <INPUTFORM TYPE="UPLOAD" WIDTH="36" REQUIRED="NO" LINKTEXT="Image"/>
</PROPERTY>
```

> **Note:**
>
> The size of a file that you can select in an upload field is limited to 30 megabytes.

Example 5-4   Creating an Upload Field with a Text Box

To create an upload field with a text box that you can enter the text in (rather than with a **Browse** button that you use to select a file), code the PROPERTY statement as follows:

1. The string set for PROPERTY NAME must begin with the letters url.

2. The value for STORAGE TYPE must be set to VARCHAR.

3. The value for INPUT TYPE must be set to TEXTAREA.

The following code snippet creates a text area field for a url column:

```
<PROPERTY NAME="urlbody" DESCRIPTION="Article Body">
  <STORAGE TYPE="VARCHAR" LENGTH="256"/>
  <INPUTFORM TYPE="TEXTAREA" WIDTH="400" HEIGHT="100" REQUIRED="YES"/>
</PROPERTY>
```

Example 5-5   Creating an Upload Field with a CKEditor

To create a CKEditor field, code the property statement as follows:

1. The PROPERTY NAME must begin with the letters url. Therefore, use a URL column for the field, otherwise the field will be too small.

2. The value for STORAGE TYPE must be set to VARCHAR.

3. The value for INPUT TYPE must be set to CKEDITOR.

```
<PROPERTY NAME="urlbody" DESCRIPTION="Body">
  <STORAGE TYPE="VARCHAR" LENGTH="50"/>
  <INPUTFORM TYPE="CKEDITOR" WIDTH="300" HEIGHT="300" REQUIRED="YES"
   INSTRUCTION="Be concise! No more than 3 paragraphs."/>
</PROPERTY>
```

The length of the type VARCHAR is the length of the path to the file where the data is stored. This is typically less than 100. An excessively large length (for example, 5000) causes issues with data storage as the database will store the data as a CLOB. And the URL field will be corrupted because the data would be stored in the database rather than the file.

Example 5-6   *Creating an Upload Field That Uploads a Binary File*

The following code creates a field where you can upload a blob. Not specifying MIMETYPE will prevent you from viewing the blob from the **Edit** and **Inspect** forms.

Code the property statements as follows:

- The string set for PROPERTY NAME must not begin with the letters url.

- The value for STORAGE TYPE must be set to BINARY.

```
<PROPERTY NAME="type_binary" DESCRIPTION="Binary">
  <STORAGE TYPE="BINARY"/>
  <INPUTFORM TYPE="UPLOAD"
   WIDTH="24" MAXLENGTH="64"
   MIMETYPE="application/msword"
   LINKTEXT="Edit with Microsoft Word"
   INSTRUCTION="maps to cc.blob"/>
</PROPERTY>
```

Consider the following about Upload fields:

- For an Upload field whose PROPERTY NAME starts with the letters **url**, only the file system path of the uploaded file is stored in the database. So, in this case, STORAGE TYPE must be set to VARCHAR.

- For an Upload field whose PROPERTY NAME doesn't start with the letters **url**, the uploaded file's data resides in the database in BINARY format. So, in this case STORAGE TYPE must be set to BINARY.

Example 5-7   *Enabling path, filename, startdate, and enddate*

The path, filename, startdate, and enddate columns are special cases.

AssetMaker creates columns for path, filename, startdate, and enddate without requiring a PROPERTY statement for them. However, while these columns exist, their fields do not appear on your asset forms unless you include a PROPERTY statement for them in the asset descriptor file.

Note the following about these columns:

- path: The STORAGE TYPE must be set to VARCHAR and LENGTH must be set to 255.

- filename: The STORAGE TYPE must be set to VARCHAR and LENGTH must be set to 128.

- startdate: The STORAGE TYPE must be set to TIMESTAMP.

- enddate: The STORAGE TYPE must be set to TIMESTAMP.

> **Note:**
>
> If you include one of these standard columns in your asset descriptor file but your storage type does not match the one specified in this list, AssetMaker cannot create the asset type.

For example:

```
<PROPERTY NAME="path" DESCRIPTION="Path">
  <STORAGE TYPE="VARCHAR" LENGTH="255"/>
  <INPUTFORM DESCRIPTION="Path" TYPE="TEXT" LENGTH="255"/>
</PROPERTY>
```

Example 5-8   Using a Query to Obtain Options for a Drop-Down List

When the INPUTFORM TYPE of your property is SELECT, you can have WebCenter Sites populate the drop-down list for the select field with a static list of items that you provide with the OPTIONDESCRIPTIONS parameter, or with a list of items that WebCenter Sites obtains, dynamically, from a database table.

Another example of a select field that populates its drop-down list dynamically from a table is the **Mimetype** field on the imagefile forms, which queries the MimeType table for its options. Here's the code:

```
<PROPERTY NAME="mimetype" DESCRIPTION="Mimetype">

  <STORAGE TYPE="VARCHAR" LENGTH="36"/>
  <INPUTFORM TYPE="SELECT" SOURCETYPE="TABLE"
   TABLENAME="mimetype" OPTIONDESCKEY="description"
   OPTIONVALUEKEY="mimetype" SQL="SELECT mimetype, description
   FROM mimetype
   WHERE keyword = 'image'
   AND isdefault = 'y'"
   INSTRUCTION="Add more options to mimetype table
   with isdefault=y and keyword=image"/>

  <SEARCHFORM DESCRIPTION="Mimetype" TYPE="SELECT"
   SOURCETYPE="TABLE" TABLENAME="mimetype"
   OPTIONDESCKEY="description" OPTIONVALUEKEY="mimetype"
   SQL="SELECT mimetype, description
   FROM mimetype
   WHERE keyword ='image'
   AND isdefault = 'y'"/>

</PROPERTY>
```

This example shows a field that not only selects items from a database table, but, through an additional SQL query, further restricts which items are returned from that table, as well.

Example 5-9   Using a Query to Obtain Labels for Radio Buttons

Use RADIO as the INPUTFORM TYPE of your property to input the label for each radio button using a static list of items that you provide with the OPTIONDESCRIPTIONS parameter, or with a list of items that WebCenter Sites obtains, dynamically, from a database table.

The following sample code creates radio buttons with labels drawn from the CreditCard table:

```
<PROPERTY NAME="sqlrbcc" DESCRIPTION="SQL RB Credit Card">
  <STORAGE TYPE="VARCHAR" LENGTH="4"/>
  <INPUTFORM TYPE="RADIO" SOURCETYPE="TABLE" TABLENAME="CreditCard"
   RBVALUEKEY="ccvalue" RBDESCKEY="ccdescription" />
  <SEARCHFORM TYPE="SELECT" DESCRIPTION="Credit Card:" SOURCETYPE="TABLE"
   TABLENAME="CreditCard" OPTIONVALUEKEY="ccvalue" OPTIONDESCKEY="ccdescription"
   SQL="SELECT ccvalue,ccdescription
   FROM CreditCard ORDER BY ccdescription"/>
</PROPERTY>
```

Example 5-10   Creating a Field with the ELEMENT Input Type

To display the fields as you want them on your asset forms, you can call custom code using the ELEMENT input type. Use this method to create asset fields, or to change the appearance of standard asset fields, though you cannot modify the storage type of a standard asset field.

An ELEMENT field can have any storage type, including BLOB. When you set a field's input type to ELEMENT, WebCenter Sites calls a custom element to display the field. The custom element must be found at one of the following locations:

For a field on the ContentForm form:

```
OpenMarket/Xcelerate/AssetType/myAssetType/ContentForm/fieldname
```

For a field on the ContentDetails form:

```
OpenMarket/Xcelerate/AssetType/myAssetType/ContentDetails/fieldname
```

For a field on the SearchForm form:

```
OpenMarket/Xcelerate/AssetType/myAssetType/SearchForm/fieldname
```

Note that *myAssetType* is the asset type that you are creating the custom field for, and fieldname is the name of the custom field.

The following excerpt from an asset descriptor file uses the ELEMENT input type:

```
<PROPERTY NAME="imagedate" DESCRIPTION="Image date">
  <STORAGE TYPE="TIMESTAMP" LENGTH="8"/>
  <INPUTFORM TYPE="ELEMENT" WIDTH="24" MAXLENGTH="48" REQUIRED="NO" DEFAULT=""
   INSTRUCTION="Format: yyyy-mm-dd hh:mm"/>
  <SEARCHFORM DESCRIPTION="Image date" TYPE="ELEMENT"
   WIDTH="48" MAXLENGTH="128"/>
</PROPERTY>
```

Note that the input form uses a customized field, but the search form and content details forms display default fields.

The following code excerpt is the element that the descriptor file calls:

```
<!-- OpenMarket/Xcelerate/AssetType/ImageFile/ContentForm/imagedate
-
- INPUT
- Variables.AssetType
Variables.fieldname
Variables.fieldvalue- default or value for this field
Variables.datatype - from STORAGE tag in ADF for this field
Variables.helpimage - help icon
Variables.alttext - help text from INPUT tag in ADF
Other fields from input tag are in:
Variables.assetmaker/property/Variables.fieldname/inputform/[tag attribute]
- field name used in form should be Variables.AssetType:Variables.fieldname
```

```
- OUTPUT
-
-->
Enter date in the format yyyy-mm-dd hh:mm:ss<br/>

<setvar
 NAME="inputfieldsize"
 VALUE="Variables.assetmaker/property/Variables.fieldname/inputform/width"/>

<callelement NAME="OpenMarket/Xcelerate/Scripts/FormatDate"/>

<INPUT TYPE="text" SIZE="Variables.inputfieldsize"
 NAME="Variables.AssetType:Variables.fieldname"
 VALUE="Variables.fieldvalue"
 REPLACEALL="Variables.inputfieldsize,Variables.fieldvalue,Variables.fieldname,
 Variables.AssetType"onChange="padDate(this.form.elements['Variables.AssetType:
 Variables.fieldname'].value,this,'Variables.AssetType:Variables.fieldname'
 );"/>

</FTCS>
```

Note that you can customize as many fields as you want using the ELEMENT input type, but you must write a separate element for each field.

5.3.2 Uploading the Asset Descriptor File to WebCenter Sites

After you have coded the asset descriptor file for your asset type, use AssetMaker to upload it and register the new elements:

1. Open the Admin interface.

2. Select the **General Admin** tree and then expand the **Admin** node.

3. Expand **AssetMaker** and click **Add New**.

   The Add New AssetMaker Asset Type form opens.

4. In the **Name** field, enter the name of the new asset type. The string that you enter into this field must exactly match the string specified by the ASSET NAME parameter in the asset descriptor file that you are going to upload.

5. In the **Descriptor File** field, click **Browse** and select the asset descriptor file.

6. Click **Save**.

   AssetMaker enters the file into the AssetType table (that is, it uploads the asset descriptor file to the default storage directory for the AssetType table), and then opens the Asset Type form.

   The Asset Type form opens.

5.3.3 Creating the Asset Table

This step continues from step 6 of Uploading the Asset Descriptor File to WebCenter Sites.

1. Select **Create Asset Table**.

2. In the **DefDir** field, examine the value and change it if necessary. AssetMaker reads this value from the asset descriptor file. You must enter a value in this field if either of the following conditions exist:

   • If you did not provide a value with the DEFDIR parameter for the ASSET tag in the asset descriptor.

   • To change the default storage directory, which is typical when you are migrating the asset type to another system.

   Enable the **Add General Category** option. AssetMaker adds one row to the Category table for the new asset type and names that category **General**.

3. Click **Create Asset Table**.

   AssetMaker creates the table, and it displays a confirmation.

4. Select **Register Asset Elements**, and then click **Register Asset Elements**.

   AssetMaker copies the elements from the `AssetStubElementCatalog` table to the asset type's directory in the `ElementCatalog` table and copies the SQL statements in the `SystemSQL` table.

   When it is finished, it displays a confirmation.

5.3.4 Configuring the Asset Type

When AssetMaker created the new asset type (in Uploading the Asset Descriptor File to WebCenter Sites), it also created an icon and administrative forms for configuring the new asset type, located in the **Admin** node on the **General Admin** tree.

To configure the asset type:

1. In the **General Admin** tree, expand the **Admin** node and then expand the **Asset Types** icon.

2. Under **Asset Types**, select your new asset type. If you do not see it in the list, right-click and choose **Refresh** from the context menu, and then choose your new asset type.

3. Click **Edit**.

   The Edit Asset Type form opens.

Figure 5-3   Edit Asset Type Form

4. (Optional) In the **Description** field, change the value, if necessary. The text in this field is the name that WebCenter Sites uses for this asset type on the forms and lists in the WebCenter Sites interface. By default, description is set to the value of the `ASSET DESCRIPTION` statement in the asset descriptor file.

5. (Optional) In the **Plural Form** field, change the value, if necessary. The text in this field is the text that WebCenter Sites uses in the Admin interface when it is appropriate to refer to the asset type in the plural. By default, Plural Form is set to the value of the `ASSET DESCRIPTION` statement in the asset descriptor file plus the letter "s." You can also specify your own plural form in the asset descriptor file by setting the value of the `ASSET` tag's `PLURAL` parameter.

6. (Optional) In the **Can Be a Child Asset** field, change the value, if necessary. By default, this field is set to `True`, which means that this asset type can be the child asset type in an association field for another asset type. Its name displays in the list of asset types in the **Child Asset Field** on the Add New Association forms.

7. (Optional) In the **Use Dimensions** field, change the value, if necessary. By default, this field is set to `True`, which means that this asset type supports the creation of multi-lingual content.

8. Click **Save**.

5.3.5 Enabling the Asset Type on Your Site

Before you can examine the forms that the new elements render for the asset type, you must enable the asset on the site (or sites) that you are working with and create a simple start menu item for it.

For instructions on how to enable your asset types and create start menu items for them, see *Oracle Fusion Middleware Administering Oracle WebCenter Sites*.

5.3.6 Fine-Tuning the Asset Descriptor File

Create a new asset of your new type and examine the New, Edit, Inspect, and Search forms. (To create a new asset of this type, click **New** on the toolbar and select the start menu shortcut that you created in the preceding procedure.)

After you examine the forms, you might have to modify the asset descriptor file.

You can make any of the following changes with relatively few steps:

- Re-ordering the fields that appear on the WebCenter Sites forms for the asset type.

- Changing the name of a field (that is, the value of `PROPERTY DESCRIPTION`).

- Changing anything in an `INPUTFORM`, `SEARCHFORM`, or `SEARCHRESULTS` statement.

To make any of the changes in the preceding list, complete the following steps:

1. Use Explorer to open and modify the asset descriptor file that you uploaded in Uploading the Asset Descriptor File to WebCenter Sites.

2. Save your changes.

3. Re-register the elements for the asset type.

You cannot change the schema of the asset type's database table after it is created. The following are the schema changes:

- Changing the name of a column (the `NAME` parameter in an existing `PROPERTY` statement).

- Changing the data type of the column (the `STORAGE TYPE` or `LENGTH` in an existing `PROPERTY` statement).

- Adding a new property (new `PROPERTY` statement), which adds a new column to the table.

- Deleting a property (an existing `PROPERTY` statement), which deletes a column from the table.

Therefore, to make any of the changes in the preceding list, you must first delete the asset type, modify the asset descriptor file, and then create the asset type again.

Note that customizing the elements that AssetMaker copied from the `AssetStubElementCatalog` table to the asset type's directory in the `ElementCatalog` table allows for changes to be overwritten at re-registration of elements.

5.3.7 Customizing the Asset Type Elements (Optional)

The following are the ways to customize asset type elements on your content management system:

- For changes specific to a certain asset type, modify the elements for that asset type in the `ElementCatalog` table, using Explorer. See About PreUpdate and PostUpdate Elements.

- To make the same modifications for assets of all types, modify the source elements in the `AssetStubElementCatalog` table, using Explorer, before you create your asset types. That way you will have to customize only once.

Note:

Although customizing source elements in the `AssetStubElementCatalog` table might be necessary, it is not supported. If you plan to customize a stub element, consider that it will be overwritten under the following conditions:

- Changing the stub elements requires you to re-register all of the asset elements that must take the new changes. When you re-register asset elements, AssetMaker moves new copies of the elements from the `AssetStubElementCatalog` table to the asset type's directory in the `ElementCatalog` table (the path is `ElementCatalog\OpenMarket\Xcelerate\AssetType`) and in the process overwrites the existing elements. Elements that you customized in the `AssetType` directory will be overwritten.

- The WebCenter Sites upgrade process typically installs new source elements in the `AssetStubElementCatalog` table. Code changes in the stub elements are overwritten when you re-register your asset types after upgrade.

Be meticulous about tracking all of your customizations at all times. That way you can re-create your work if necessary.

For more information about the asset type elements you can modify, see The Elements.

5.3.8 Adding Subtypes (Optional)

A subtype is a subclass of the basic asset type based on how that asset is rendered. Use subtypes to define a separate default approval template for an asset of that type and subtype on each publishing target.

Follow these steps to create a subtype:

1. In the **General Admin** tree, expand the **Admin** node, and then expand the **Asset Types** option.

2. Under the **Asset Types** option, select the asset type that you want to create subtypes for.

3. Select the **Subtypes** option.

   The Subtypes form opens.

Figure 5-4   Subtypes for Asset Type Dialog

4. Click **Add New Subtype**.

5. In the **Name** column of the next form, enter the name of the subtype.

6. In the corresponding field of the **Sites** column, select the names of the sites that need this subtype.

7. Repeat these steps for up to five subtypes.

8. Click **Save**.

5.3.9 Configuring Association Fields (Optional)

Associations are assettype-specific relationships that describe parent-child relationships or links between individual assets or asset sub-types. Associations are described in The Basic Asset Model.

When you code your template elements, you use the names of these relationships to identify individual assets or sub-types, without having to refer to the object by its name. For example, in a site that associates imagefile assets with article assets, the template elements should be coded to extract an associated imagefile asset by the name of the association rather than the name of the asset. This enables the template to display another imagefile for an article without you having to re-code the template.

Associations are represented as fields in the asset forms. These fields are not created from an asset descriptor file. Instead, you use the Asset Associations forms under the **Admin** node in the **General Admin** tree.

To add an association field:

1. In the **General Admin** tree, expand the **Admin** node, and then expand **Asset Types**.

2. Under the **Asset Types** option, select the asset type that you want to create associations for.

3. Under the asset type you selected, select **Asset Associations** and then **Add New**.

   The Add New Association form opens.

   Figure 5-5   Add New Association Dialog

4. In the **Name** field, enter the name of the association field, without using spaces, decimal points, or punctuation marks.

5. In the **Description** field, enter a short description of the field. WebCenter Sites uses the text entered into this field as the name of the field when it is displayed on the new asset form.

6. In the **Child Asset** field, select the kind of asset type that will appear in this field. (It is called the **Child Asset** field because associations create parent-child relationships between assets.) You cannot specify the template or the page asset type in this field.

7. In the **Subtypes** field, select the subtype or subtypes for this association by highlighting them. To select multiple subtypes, press the **Control** key while you click your selection with your mouse.

8. In the **Mirror Dependency Type** section, select a suitable option for this asset association. By default, it is set to **Exists**. The dependency type specified here is used by the approval system when your publishing method is Mirror to Server.

9. Click **Add New Association**.

   WebCenter Sites creates a row in the `Association` table for this association. The name used in the row is the text you entered in the **Name** field in step 4.

5.3.10 Configuring Categories (Optional)

Use categories to organize your asset types according to some convention that works for your site design. Although all basic asset types have a **Category** field (column) by default, it is not a required field and you do not have to use it.

To add a category:

1. Under the **Admin** node, select **Asset Types.**

2. For the **Asset Type** option, select the asset type for which you want to create categories. For example, select **Article**.

3. Under that asset type, select **Categories** and then **Add New**.

   The Add New Category form opens.

   Figure 5-6   Add New Category Dialog

4. In the **Description** field, enter a short description of the category. WebCenter Sites uses the text that you enter in this field in the site tree and in the drop-down list for the **Category** field on the forms for assets of this type.

5. In the **Category Code** field, enter a two-character code for your new category.

6. Click **Add**.

7. Repeat steps 2 through 6, as needed, to finish creating the categories for this asset type.

The categories you created now appear in the drop-down lists in the **Category** fields when creating new assets or editing assets.

5.3.11 Adding Mimetypes (Conditional)

The `MimeType` table holds mimetype codes that can be displayed in **MimeType** fields. You must add mimetypes for your asset if you reference the `MimeType` table in your asset descriptor file. For example, both the imagefile and stylesheet asset types have upload fields with **Browse** buttons next to them. After you select a file in the upload field, you specify the mimetype of the file you selected from the **Mimetype** drop-down list.

The **Mimetype** fields for the imagefile and stylesheet asset types query the `MimeType` table for mimetype codes based on the `keyword` column:

- Mimetype codes with their `keyword` set to `stylesheet` appear in the drop-down list of the **Mimetype** field in the Stylesheet form.

- Mimetype codes with their `keyword` set to `image` appear in the drop-down list of the **Mimetype** field in the **ImageFile** form.

By default, stylesheet files can be `CSS` files and imagefile files can be `GIF` or `JPEG` files. You can add mimetype codes for these asset types, for your own custom asset types, or for any other reason.

To add mime types to a **Mimetype** drop-down list:

1. Open Explorer.

2. Expand the `MimeType` table.

3. Do one of the following:

    - To add a mimetype for the imagefile asset type, select `image` in the `MimeType` table.

    - To add a mimetype for the stylesheet asset type, select `text` in the `MimeType` table.

    - To add a mimetype for a custom asset type with an upload field or for any other reason, select the appropriate location in the `MimeType` table.

4. Right-click in the frame on the right and then choose **New** from the drop-down list.

    Explorer creates a new row in the table.

5. In the **mimetype** field, enter the name of the mimetype. For example: `XSL`.

6. In the **extension** field, enter the extension for mime types of this type. For example: `.xml`.

7. In the **Description** field, enter a short description of this mimetype.

8. In the **isdefault** field, do one of the following:

    - To specify multiple extensions for the same mimetype, enter `n`. For example, if a mimetype named JPG has `.jpg` and `.jpeg` extensions, set isdefault to `n`.

    - If this is the only extension for the mimetype, enter `y`.

9. In the **Keyword** field, do one of the following:

- To add a mimetype for the imagefile asset type, enter `image`.

- To add a mimetype for the stylesheet asset type, enter `stylesheet`.

- To add a mimetype for a custom asset type with an upload field or for any other reason, enter the appropriate keyword.

10. Select **File** and then **Save all**.

Explorer saves the row.

A mimetype code, if added with the keyword of `image`, is now displayed in the **Mimetype** field of the ImageFile form. A mimetype code that was added with the keyword of stylesheet is now displayed in the **Mimetype** field of the Stylesheet form.

5.3.12 Editing Search Elements to Enable Indexed Search (Optional)

WebCenter Sites and Oracle WebCenter Sites: Engage have its own database SQL search mechanism that runs the Simple and Advanced searches. However, you can set up your management system to one of the supported third-party search engines instead. For configuration information, see *Oracle Fusion Middleware Administering Oracle WebCenter Sites*.

When you are using a search engine on your management system, each asset is indexed when it is saved after being created or edited. By default, only the default fields are indexed (for a list, see Default Columns in the Basic Asset Type Database Table). To index the fields that you created with `PROPERTY` statements in your asset descriptor file, add statements for them in the following elements:

- `OpenMarketXcelerate/AssetType/YourAssetType/IndexAdd.xml`

- `OpenMarketXcelerate/AssetType/YourAssetType/IndexReplace.xml`

To add the asset type's custom fields to these elements, use the WebCenter Sites `INDEX` tags and follow the convention illustrated in these elements.

5.3.13 Creating and Assigning Asset Type Icons (Contributor Interface Only)

Create and assign the icon that will represent the asset type in the Contributor interface's navigation trees (**Site Tree**, **Content Tree** and **My Work** tree). Icons can be of any type of image file (for example, PNG, GIF, and so on). In this example, an image file of type PNG is created.

1. Create an image no larger than 20x20 pixels representing the asset type.

2. Name the file using the syntax `assetType.png`. The file name determines the asset type for which the icon will be displayed. The name is case-sensitive.

3. Place the file in the following directory:

`<cs_app_dir>/Xcelerate/OMTree/TreeImages/AssetTypes/`

where `<cs_app_dir>` is the directory of the deployed WebCenter Sites application on your application server.

4. Restart your application server for the icons to appear in the Contributor interface (**Site** tree, **Content** tree, and **My Work** tree).

5.3.14 Coding Templates for the Asset Type

Creating your asset types and coding the templates for assets of that types is an iterative process.

- Although you have to create asset types before you can create templates for assets of that type, it is likely that you will discover areas that need refinement in your data design only after you have coded a template and tested the code.

  See Coding Elements for Templates and CSElements.

5.3.15 Moving the Asset Types to Other Systems

When you have finished creating all of your new asset types (including creating templates for them), you need to move them to other systems.

- Migrate the asset types to the management and delivery systems.

  System administrators will then configure the asset types for the management system. They will enable revision tracking where appropriate, create workflow processes, create start menu shortcuts, and so on.

  See *Oracle Fusion Middleware Administering Oracle WebCenter Sites*.

5.4 Deleting Basic Asset Types

You can either delete components of the asset type that you select or all traces of the asset type from the database.

The asset type components are:

- The database table and all the data in it.

- The elements in the `ElementCatalog` table.

- The SQL statements in the `SystemSQL` table.

- The row in the `AssetType` table.

- Any rows in the `Association` table (optional).

- Any rows in the `Category` table (optional).

- Any rows in the `AssetPublication` table.

- Any rows in the `AssetRelationTree` table.

To delete an asset type that you created with AssetMaker:

1. Open the Admin interface.

2. Select the **General Admin** tree and then expand the **Admin** node.

3. Expand **AssetMaker** and then select the asset type that you want to delete.

   The Asset Type form opens.

4. Select the **Delete Asset** option and click **Submit**.

   WebCenter Sites displays a confirmation message.

5. Click **OK**.

6

Designing Flex Asset Types

Flex asset types are part of a flex asset family, and they inherit attributes from parents, grandparents, and so on. Each asset type is stored across several database tables.

For an overview of the data model, see Understanding the Asset Types and Asset Models. For information about flex asset types, see these topics:

- About Designing Flex Asset Types

- Design Tips for Flex Families

- The Flex Family Maker Utility

- Flex Asset Elements

- Setting Up Your Development System

- Creating a Flex Asset Family

- What You May Need to Know About Editing Flex Attributes, Parents, and Definitions

- Using Product Sets

6.1 About Designing Flex Asset Types

When creating flex asset types, you also create the individual data structure assets of those types. These assets are flex attributes, flex parent definitions, flex definitions, and flex parent assets. For creating flex asset types, the Flex Family Maker utility is available in the Admin interface.

Typically, you design the flex asset types and create the data structure assets on a development system. When your data model is ready, you migrate your work from the development system to the management and delivery systems.

6.2 Design Tips for Flex Families

The data structure that you create for your flex family should offer flexibility and convenience to the site visitors and content contributors who enter data into the WebCenter Sites database. Here are some tips that can help you meet the site visitors' and content teams' needs.

See these topics:

- Visitors on the Delivery System

- Users on the Management System

- How Many Attribute Types Should You Create?

- Designing Flex Attributes

- [How Many Definition Types Should You Create?](#)

- [Designing Parent Definition and Flex Definition Assets](#)

6.2.1 Visitors on the Delivery System

The experience of the visitors to your online site is based on the following asset types:

- Flex asset

- Flex attribute

Your online site pages display flex assets (assetsets) for the visitors through queries that are based on attribute values (searchstates). To give the appearance of hierarchy on your online site, use attribute values as the basis for drill-down searches.

6.2.2 Users on the Management System

The users of your management system navigate through a visual hierarchical structure that you create for them with the following flex asset types:

- Flex parent definition

- Flex definition

- Flex parent

Although the organizational structure that you create with these asset types does affect the data, it determines which attribute values are inherited by which flex assets. Its biggest impact is on the users of the management system.

You are not required to use flex parents and flex parent definitions, but their inheritance properties make them a valuable tool for users who are maintaining a large amount of data such as an online catalog:

- Changing an attribute value at the parent level changes that value for all the flex assets who are children of that parent, which means you only have to change the value once.

- Inherited attribute values aren't subject to user error, which means less data cleanup is required.

The inheritance tree that you create for your content providers has no bearing on how your site visitors navigate the online site you are designing. For example, if content is entered into your management system through some completely automated process (perhaps it is bulk loaded from an ERP system) you would have no need for parent asset types at all, yet you can still create drill-down searches on your online site.

6.2.3 How Many Attribute Types Should You Create?

As described in Assetsets and Searchstates, only the flex assets that share a common attribute type can belong to the same assetset because queries (searchstates) are based on attributes and not on the organizational constructs of parent definitions and flex definitions.

You might create a nicely delineated interface on the management system by organizing you data to use separate types of attributes, but this data cannot be synthesized well on the delivery system. As a general rule, you should create one type of attribute for your system. Multiple versions of the rest of the family members (the flex asset type, flex definition type, flex parent type, and flex parent definition type)

must still share the same pool of attributes. For example, in the avisports sample site, the article asset type and the image asset type share the same attribute type. Therefore, you are able to create assetsets that contain an article and a corresponding image for that article.

6.2.4 Designing Flex Attributes

Before you begin creating attributes, design them on paper. Determine all the attributes you need and decide where they will appear, with flex assets or the flex parents. Start by planning out the bottom level of your hierarchy (that is, the individual instances of flex asset types like products) and determine the attributes you need for each item at that level. For example, before creating flex filter assets, determine which attributes must be created and assigned to the definitions as the input and output attributes for your filters.

You must determine all of the flex attributes that you need beforehand because the way you plan to use these attributes creates dependencies that you must account for when you create them.

6.2.4.1 Which Data Types Are Available for Attributes

Assess the data types that are available for attributes and the default input types for those data types. Determine which data types will work best for which attributes. To change the default input style for an attribute, you create an attribute editor for it before you create the attribute. See Designing Attribute Editors.

When you create a flex asset that uses an attribute of type `blob`, the format of the value entered for the attribute on an Inspect form depends on its type. For example, a text file shows the first 200 bytes in the file. An image file displays as a thumbnail image. And some files cannot be displayed at all. In this case, WebCenter Sites displays the message **filename not displayable** but the file location is still successfully recorded.

6.2.4.2 About Using Attribute Editors

The default input type for an attribute depends on the data type that you select for it. You can create an attribute editor instead of using the default input type.

Creating flex assets and their attribute editors is an iterative process. You can create the attribute editors first or the attributes first and then go back and assign the attribute editors. For information about process of creating attribute editors, see Designing Attribute Editors.

About Attributes of Type Blob

The default input style of an attribute of type `blob` is a text field with a **Browse** button. Click **Browse** to locate and select a file and WebCenter Sites uploads it to the default storage directory. You cannot use the WebCenter Sites forms to edit the contents of the file.

To be able to enter content directly into the external file through the WebCenter Sites forms, you must assign an attribute editor to the attribute:

- For an attribute editor that uses the `TEXTAREA` input style, create a field that can hold up to 2,000 characters (entered through the forms). When saved, that content is written to the default storage directory.

- In your CKEditor, you can use a **CKEDITOR** field to edit the contents of the external file that the attribute represents.

About Attributes of Type Asset

The default input style for an attribute of type asset is a drop-down list of all the assets of the type specified. An unfiltered drop-down list is not recommended for more than 20 assets of that type.

In general, whenever you create an attribute of type asset, you should assign it an attribute editor.

- An attribute editor that uses the `PICKASSET` style checks to find out whether the tree is toggled on or off in the WebCenter Sites interface. If the tree is on, the user can select an asset from a tab in the tree. If the tree is toggled off, the attribute editor displays a dialog that lists the assets from the **Bookmarks** and **History** tabs.

- Another option is to use the `PULLDOWN` style but to supply a query asset that limits the options that appear in the list.

- For valid choices of assets that are small in number, use the `CHECKBOXES` or the `RADIOBUTTONS` input style, both of which require a query asset to identify the assets.

6.2.4.3 Where Will Each Attribute Be Used?

After you have determined the list of attributes, determine whether you plan to use them in a flex definition or a flex parent definition. Sort them logically by using the following guidelines:

- An attribute whose value is unique to an individual flex asset (product, article, image, for example) belongs at the bottom of the tree, with the flex asset.

- An attribute whose value is the same for multiple flex assets belongs in a parent. Of course there are always exceptions. For example, a toaster that costs the same amount as a bowling ball is unlikely to inherit its prices from a common parent.

- Based on that attribute distribution, you can determine how many flex definitions you need and how many parent definitions you need.

Remember that there is both a physical limit (based on your DBMS) and a psychological limit (user satisfaction) as to how many attributes you can or should use in an individual flex asset or flex parent. Someone has to enter all those values. Be sure to create and then assign to the definitions only those attributes that you really plan to use. You can add more attributes when you need additional ones.

6.2.4.4 Attribute Dependencies Imposed by Hierarchy

After you know where an attribute will be used, you can determine whether hierarchical concerns add requirements to the attribute. For example, an attribute used by a flex parent must be configured to hold multiple values for data structure that allows flex assets to have multiple parents, because a flex asset might inherit multiple values for it. In general, try not to make the inheritance structure too complex.

6.2.5 How Many Definition Types Should You Create?

The appearance and input of data on the management system is based on the flex asset definitions and the flex parent definitions. Parents and flex assets appear on tabs in the tree based on the hierarchy that you create through the definitions.

In general, it is best to create a separate set of definition types for each flex asset member in a family. For example, in the avisports sample site, the article and image flex assets share the same attribute asset type, but they have different parents, and flex

definitions. The definition for article assets contains only attributes that are relevant to articles, whereas, the definition for image assets contains only attributes that are relevant to image assets. Article and image assets that share the same parents and definitions will have some attributes left blank in both types of assets because some attributes won't apply to article assets and some to image assets.

6.2.6 Designing Parent Definition and Flex Definition Assets

The hierarchy on the tabs in the tree in the WebCenter Sites interfaces is created through the flex parent definitions and flex definitions:

- To set a hierarchy three levels deep, you need at least two parent definitions and at least one flex definition.

- To specify a hierarchy two levels deep, you need at least one parent definition and at least one flex definition.

Be sure to consider the basic tenets of usability when you set up a structural hierarchy with the flex definitions and flex parent definitions. For example:

- How deep can the hierarchy go before the content providers feel lost in the tree?

- How many attribute values can be inherited to alleviate the possibility of user error during input?

- How many options can be comfortably displayed in a drop-down list?

The content providers won't like to use a complex system. Keep the following rules in mind as you design the data structure with a flex family for your online site:

- Carefully planned, easy-to-use asset design (data design) makes content providers happy.

- Usable layout and efficient code makes site visitors happy.

And both user groups need efficient systems that perform well.

6.2.6.1 Determining Hierarchical Place

You can log in to WebCenter Sites, access the avisports sample site, and examine the form for a new content parent definition or for a new content definition.

In the Parent Definition section of these forms, you determine two things:

- The hierarchical position of the assets that use this definition.

- The parents that they can inherit attributes from.

Remember that although the hierarchical position has meaning only in the Oracle WebCenter Sites: Contributor interface on the management system, the attributes that they inherit have meaning both on the management system and on your online site.

The text box named **Available** lists all the existing parent definitions. You use this section of the form to specify how many parents are possible, by selecting parent definitions from the **Available** list and moving them to the **Selected** list.

When you create a parent asset or a flex asset, the New form displays a definition field in which you specify a definition for the parent or flex asset. The definitions available for you to select are determined by the definitions you selected from the **Available** list when you created the definition you are using to create the parent or flex asset.

> **Note:**
>
> By default, the available definitions for the parent or flex asset you are creating are displayed in a drop-down list. However, when you create the parent definition, you can specify whether this will be displayed as a drop-down list, type ahead field, or drop zone (pick from tree) field.

The asset inherits the attribute values (if any) of the parent selected in the New form. The more parent definitions you select from the **Available** list, the more fields the content providers have to fill out when they create a new flex asset. Not selecting a parent definition in the **Available** list positions the assets created with this definition at the top level of the tree on the tab that displays your flex assets. The best way to understand how parent definitions, flex definitions, parent assets, and flex assets interact is to examine the assets delivered with the avisports sample site.

6.2.6.2 Determining Attribute Inheritance

You can configure attribute inheritance in the **Attributes** section of the parent definition form. You use that section to specify the attributes that define the parents that are created with this definition. When you create a parent with this definition, the values entered for these attributes are passed down to the flex assets that are children of the parent asset.

6.2.6.3 How Many Flex Parent Definition Assets?

Consider usability when you decide how many flex parent definition assets and how many parent assets of those definitions that you need. For short drop-down lists in the new parent and new flex asset form, create many parent definitions so that there are fewer parents with each definition. However, to have a small number of parent definitions and a large number of parents, create a tab that lists all the parents so the content providers can select the correct parent asset from the tab.

6.2.6.4 How Many Flex Definition Assets?

Create enough flex definitions so that fields (attributes) are not left blank on the New and Edit flex asset forms. Creating few definitions increases the form size with lots of attribute fields, not all of which apply for each asset. When you have long forms with lots of attribute fields, not only do content providers have to sort through the form to determine which attributes apply to the asset they are currently creating, the form takes a long time to be rendered in the user's browser.

6.3 The Flex Family Maker Utility

The Flex Family Maker utility creates flex families and their database tables. It writes information to the database tables and creates directories in the ElementCatalog table to which the utility copies elements.

- Creates several database tables (the number depends on which flex asset types you create).

- Writes information about the new flex family to these tables:

  - `FlexAssetTypes`: Holds a row for each flex asset member type.

  - `FlexGrpTmplTypes`: Holds a row for each flex parent definition type.

  - `FlexGrpTypes`: Holds a row for each flex parent type.

- `FlexTmplTypes`: Holds a row for each flex definition type.

- Creates new directories in the `ElementCatalog` table using the following naming convention:

  `OpenMarket/Xcelerate/AssetType/`*`NameOfYourAssetType`*

- Copies elements from the `ElementCatalog` table to the directories created for your asset types. WebCenter Sites use these elements to format the New, Edit, Inspect, Search, and Search Results forms for assets of that type.

For information about the main database tables for flex assets and flex parent assets, see Flex Families and the Database.

6.4 Flex Asset Elements

The Flex Family Maker creates elements and SQL statements and stores them in appropriate database tables.

The Flex Family Maker copies elements for the new flex asset type to `OpenMarket/Xcelerate/AssetType/NameOfAssetType` in the `ElementCatalog` table. For example, the avisports sample site article asset elements are in:

`OpenMarket/Xcelerate/AssetType/AVIArticle`

It also creates a SQL statement that the search elements use and places it in the `SystemSQL` table under `OpenMarket/Xcelerate/AssetType/`*`NameOfAssetType`*.

For information on the elements and SQL statement that Flex Family Maker copies for you, see Elements and SQL Statements for the Asset Type. The elements for flex assets are the same as the elements for the basic assets with the exception of the `AppendSelectDetailsSE` element.

6.5 Setting Up Your Development System

Before you create a flex asset family, you need to set up your development system and get access to it.

1. Create the appropriate WebCenter Sites sites.

2. Create a user name for yourself that has administrator rights, and enable that user name on all of the sites on your development system.

   Without administrator rights, you do not have access to the **Admin** node in the **General Admin** tree, which means that you cannot perform some procedures in this chapter.

3. Assign the `Designer` and `GeneralAdmin` roles to your user name so you will have access to all the trees in the WebCenter Sites Admin interface and all of the existing **Start** menu shortcuts for the assets in the sample site.

   Be sure that the `TableEditor` ACL is assigned to your user name.

   See *Oracle Fusion Middleware Administering Oracle WebCenter Sites*.

6.6 Creating a Flex Asset Family

For your flex asset data model you need to create flex asset types and individual data structure assets of those types: flex attributes, flex parent definitions, flex definitions, and flex parent assets.

These topics are presented in the order in which a new flex asset family should be created. So, to create a flex family for the first time, follow the procedure in the sequence given here:

- Creating a Flex Family.

- (Conditional) Creating Additional Flex Family Members.

- (Conditional) Configuring the Flex Family Members.

- Enabling the New Flex Asset Types on all the WebCenter Sites sites on the development system and create start menu shortcuts for all the new asset types.

- Create Flex Attributes and design your attribute editors. See Designing Attribute Editors.

- (Conditional) Creating Flex Filter Assets . See Creating Flex Filters.

- Creating Parent Definition Assets.

- Creating Flex Definition Assets.

- Creating Flex Parent Assets.

- Creating and Assigning Asset Type Icons (Contributor Interface Only) that will represent the members of your flex family in search results lists that are displayed in the Thumbnail view.

- Coding Templates for the Flex Assets.

- Testing Your Design (Creating Test Flex Assets) by creating enough flex assets to examine the data structure that you have designed.

- (Conditional) Creating Flex Asset Associations.

- Moving Asset Types to Other Systems. See *Oracle Fusion Middleware Administering Oracle WebCenter Sites*.

6.6.1 Creating a Flex Family

To create a new flex family:

> **Note:**
>
> WebCenter Sites is case-insensitive to the names of the flex assets regardless of the underlying case-sensitivity of the database. If you create a flex family with the name of an existing asset but different case, WebCenter Sites adds the rest of the flex family under the existing asset type(s). For example, if the following flex family already exists {Product_A, Product_PD, Product_CD, Product_P, Product_C}, and you try to create this flex family: {PRODUCT_A, PRODUCT_PD, PRODUCT_CD, PRODUCT_P, myProduct_C}, then WebCenter Sites will use the existing asset types and create the following flex family: {Product_A, Product_PD, Product_CD, Product_P, myProduct_C}.

1. In the **General Admin** tree, select the **Admin** node.

2. Expand **Flex Family Maker** and click **Add New Family**.

   The Add New Flex Family form opens.

3. In the Add New Flex Family form, fill in the following fields to name the members of the flex family:

   a. In the **Flex Attribute** field, enter the name of the new flex attribute asset type.

   The name you enter in this field is the internal name of the new attribute asset type. It becomes the name of the core table for this asset type and the prefix for all its auxiliary tables.

   > **Note:**
   >
   > There are three suffixes reserved for use. Flex attributes cannot have a name end with _size, _type, or _file. For example, broadformsize is an acceptable flex attribute name; broadform_size is not.

   b. In the **Flex Parent Definition** field, enter the name of the new flex parent definition asset type.

   The name you enter in this field is the internal name of the new parent definition asset type. It becomes the name of the core table for this asset type and the prefix for all its auxiliary tables.

   c. In the **Flex Definition** field, enter the name of the new flex definition asset type.

   The name you enter in this field is the internal name of the new flex definition asset type. It becomes the name of the core table for this asset type and the prefix for all its auxiliary tables.

   d. In the **Flex Parent** field, enter the name of the new flex parent asset type.

   The name you enter in this field is the internal name of the new parent asset type. It becomes the name of the core table for this asset type and the prefix for all its auxiliary tables.

   e. In the **Flex Asset** field, enter the name of the new flex asset type.

The name you enter in this field is the internal name of the new flex asset type. It becomes the name of the core table for this asset type and the prefix for all its auxiliary tables.

f. In the **Flex Filter** field, enter the name of the new flex filter asset type.

The name you enter in this field is the internal name of the new flex filter asset type. It becomes the name of the core table for this asset type and the prefix for all its auxiliary tables.

4. Click **Continue**.

The Add New Flex Family form opens. WebCenter Sites automatically populates the **Description** and **Plural Form** fields.

Figure 6-1   Add New Flex Family Dialog

5. In the Add New Flex Family form, do the following:

a. To enter your own values into the **Description** field for each member of the family, enter the external name of the asset type, that is, the name of the asset

type when it is displayed in WebCenter Sites. This is the name that displays on the forms (New, Edit, Inspect, and so on).

    b. To enter your own values into the **Plural Form** field for each member of the family, enter the plural version of its name. This version is used in status messages and so on when appropriate.

6. Click **Add New Flex Family**.

Flex Family Maker creates the database tables that will store assets of these types.

It also copies elements that format the forms for assets of these types to a directory with the name of the asset type in the `ElementCatalog` and `SystemSQL` tables.

6.6.2 (Conditional) Creating Additional Flex Family Members

To create additional flex family members (for example, if you need multiple flex parent asset types):

1. In the **General Admin** tree, expand the **Admin** node, and then expand the flex family you just created.

2. Drill down the flex family tree until you reach the type of flex family member you want to create (flex parent for example).

3. Under the type of flex family member you want to create, double-click **Add New** *Member* **Asset Type**.

The New *Member* Asset Type form opens.

Figure 6-2   Flex Family Maker: New Parent Definition Asset Type Form

4. In the form, fill out the required fields and click **Save**.

WebCenter Sites displays a message confirming that the asset type was created.

5. Repeat this procedure for each additional flex family member you want to create.

6.6.3 (Conditional) Configuring the Flex Family Members

When the Flex Family Maker creates the new flex family, it also creates an icon and administrative forms for configuring the new flex family, located on the **Admin** tab.

1. In the **General Admin** tree, expand the **Admin** node, and then expand the **Asset Types** icon.

2. Under **Asset Types**, select one of the asset types in your new flex family. To see the asset types for your flex family in the list, right-click the navigation pane and choose **Refresh All** from the context menu, and then select an asset type in your new flex family.

3. Click **Edit**.

   The Edit Asset Type form opens.

4. In the Edit Asset Type form, do the following:

   a. In the **Description** field, change the value, if necessary. The text in this field is the name that WebCenter Sites uses for this asset type on the forms and lists in the WebCenter Sites interfaces. By default, **Description** is set to the value you specified in the Add New Flex Family form for that asset type.

   b. In the **Plural Form** field, change the value, if necessary. The text in this field is the text that WebCenter Sites uses in the WebCenter Sites interfaces when it is appropriate to refer to the asset type in the plural.

   c. In the **Can Be a Child Asset** field, change the value, if necessary. To allow the asset type to be the child asset type in an association field for another asset, set the value to `True`.

   d. In the **Use Dimensions** field, change the value if necessary. To enable this asset type to support the creation of multilingual content, set the value to `True`.

5. Click **Save**.

6. Repeat steps 1 to 5 for each flex family member you want to configure.

6.6.4 Enabling the New Flex Asset Types

Before you start creating assets (attributes, flex parent definitions, and so on), you must complete some steps on the **Admin** tab. Your login must grant you administrator rights to access to the **Admin** tab.

To enable the new flex asset types:

1. In the **General Admin** tree, expand the **Admin** node, and then click the **Sites** icon and complete the following steps:

   a. Expand the site that you are going to use to work with these asset type.

   b. Under that site, select **Asset Types** and then **Enable**.

   c. Select your new asset types from the list and click **Enable Asset Types**.

      WebCenter Sites can automatically create a new start menu item or a search start menu item (or both) for the asset types you are enabling.

   d. Enable the option next to any available start menu item that you would like WebCenter Sites to create, as shown in the following figure:

Figure 6-3   Start Menu Selection Dialog
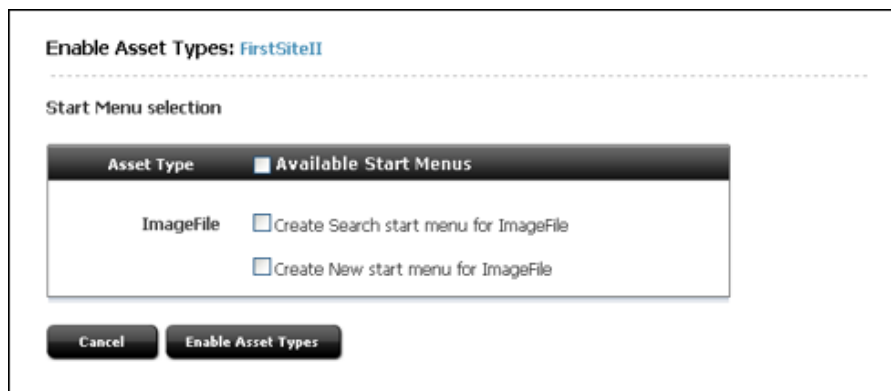
> Generate these menu items now, or you or your site administrator must manually create them later (no one can create assets of the enabled asset types until start menu items are created for them).

2. Click **Enable Asset Types**.

3. The asset types are now enabled for the site(s). If you did not use WebCenter Sites to generate start menu items, you or your site administrator must now manually create them. As the developer of the asset types and the designer of the online site, your responsibility is to let the administrator know enough about your assets and site design that the site administrator can configure meaningful start menu items.

> You (the developers) must let the site and system administrators know which fields are used by the queries, collections, or other design elements for your online site so that they can create meaningful start menu items for the content providers.

4. Repeat steps 1 through 3 for each site on which you want to enable these asset types.

After you or your administrator has created start menu items for your new asset types, you can create assets of these types.

Note:

You may have added your asset types to a tab in the tree. However, to create assets of these types you *must* create start menu shortcuts for them.

6.6.5 Create Flex Attributes

The steps can differ significantly based on the data type that you select for your attribute. Therefore, this section presents several procedures:

- Creating Flex Attributes: Basic Procedure

- Creating Flex Attributes of Type Blob (Upload Field)

- Creating Flex Attributes of Type Asset

- Creating Foreign Flex Attributes

6.6.5.1 Creating Flex Attributes: Basic Procedure

This example shows you the basic procedure for creating flex attributes.

1. In the menu bar, click **New**, and then select the name of your attribute type from the list of shortcuts. For example, **New Product Attribute**. To be able to select assignees for the workflow associated with this asset, choose at least one user from each role and click **Set Assignees**.

   The Product Attribute form opens.

   Figure 6-4   Product Attribute Form

2. In the form, fill in the following fields:

   a. In the **Name** field, enter a name of up to 64 characters, excluding spaces.

   b. In the **Description** field, enter a short, descriptive phrase that describes the use or function of the attribute.

   c. In the **Value Type** field, select a data type for this attribute:

      • If you select **blob**, see Creating Flex Attributes of Type Blob (Upload Field) and start with step 3.

      • If you select **asset**, see Creating Flex Attributes of Type Asset.

      • To create a foreign attribute, select a value other than **asset** or **blob**, complete step d, and continue with step 3 in the Creating Foreign Flex Attributes.

      • If you select **text**: The Admin interface supports searches on attribute values of all types except **text**.

      For information about deciding which data type is appropriate for your attribute, see Data Types for Attributes.

   d. In the **Number of Values** field, select either **single** or **multiple** from the drop-down list, as appropriate for the data type that you selected in the **Value Type** field.

      To make this attribute available to a flex parent (if your data structure allows multiple flex parents for a flex asset), select **multiple** because the flex assets that inherit values for this attribute might inherit a value from multiple parents.

> **Note:**
>
> When an attribute is configured to accept multiple values, it displays on the flex parent and flex asset forms as a field with an **Add Another Attribute Name** button. To enable the attribute to accept multiple values for inheritance reasons without allowing content providers to select multiple values for the attribute for individual parents or flex assets, assign the attribute an attribute editor that presents it as a single-value field (but select multiple in the **Value Type** field).
>
> WebCenter Sites does not allow attributes of type text to have multiple values, due to the way these attributes are stored in the database. A message denoting this restriction displays if you attempt to save an attribute configured in such a way.

    e. To use any other input type than the default type for this attribute (which is based on the data type that you selected in the **Value Type** field), in the **Attribute Editor** field, select an attribute editor from the drop-down list.

> **Note:**
>
> If you select the CHECKBOXES attribute editor, ensure the **Number of Values** field is set to **multiple**.

    If you need more information about:

- default input types (so you can determine whether you want to use an attribute editor instead), see Default Input Styles for Attributes.

- creating attribute editors, see Designing Attribute Editors.

- which attribute editors are appropriate for the data type of this attribute, see The Attribute Editor Asset.

    f. To override the default ISO character set (ISO 8859-1), in the **ISO Character Set** field, enter the one you want to use for this attribute.

3. Click **Save**.

6.6.5.2 Creating Flex Attributes of Type Blob (Upload Field)

This example shows you how to create a flex attribute of type Blob.

1. Complete steps 1 through 2c in Creating Flex Attributes: Basic Procedure.

2. In the **Value Type** field, select **blob**.

3. (Optional) In the **Folder** field, enter a path to the directory that you want to store the attribute values in. The value you enter in this field is appended to the value set as the default storage directory (defdir) for the MungoBlobs table.

4. In the **Number of Values** field, select **single** or **multiple**, as appropriate.

5. If you do not want to use the default input type (a **Browse** button), in the **Attribute Editor** field and select one of the following:

- An attribute editor that specifies the TEXTAREA input style.

- If your system is configured to use CKEditor, an attribute editor that specifies the CKEDITOR input style.

6. Click **Save**.

6.6.5.3 Creating Flex Attributes of Type Asset

To create an attribute of type asset:

1. Complete steps 1 through 22.b in Creating Flex Attributes: Basic Procedure.

2. In the **Value Type** field, select **asset**.

3. In the **Asset Type** field, select an option from the drop-down list.

4. In the **Mirror Dependency Type** field, select a dependency type.

5. In the **Number of Values** field, select either **single** or **multiple** from the drop-down list, as appropriate for the data type that you selected in the **Value Type** field.

If this attribute is to be used by a flex parent and your data structure allows flex assets to have multiple flex parents, you must select **multiple** because the flex assets who inherit values for this attribute might inherit a value from multiple parents.

6. If the number of assets of the type you selected in the **Number of Values** field is more than 20, select one in the **Attribute Editor** field. For information about appropriate attribute editors, see About Using Attribute Editors.

7. Click **Save**.

6.6.5.4 Creating Foreign Flex Attributes

To keep data in another system (a price list, for example) that you also want to use for your flex assets, create a foreign attribute that points to the column in the foreign table whose data you want to use as a flex attribute.

To create a foreign attribute:

1. Register the foreign table that contains the data you want to use as a flex attribute.

2. Complete steps 1 to 2d in Creating Flex Attributes: Basic Procedure.

> **Note:**
>
> In the **Value Type** field, you cannot select either **asset** or **blob**.

3. If you plan to use WebCenter Sites flex asset forms to enter values for the attribute into the foreign table and you do not want to use the default input type for the data type that you selected in the **Value Type** field, in the **Attribute Editor** field, select an appropriate attribute editor.

4. In the **Editing Style** field, do one of the following:

- To use WebCenter Sites forms to enter values into this attribute's fields for the flex assets that use it, select **local**.

- If you do not want users to be able to write values to this table through WebCenter Sites forms, select **external**.

5. In the **Storage Style** field, select **external** from the drop-down list.

6. In the **External ID** field, specify the name of the column that serves as the primary key for the table that holds this foreign attribute, that is, the column that uniquely identifies the attribute.

7. In the **External Table** field, enter the name of the table that stores this attribute.

8. In the **External Column** field, enter the name of the column in the table specified in the External Table that holds the values for this attribute.

9. Click **Save**.

6.6.6 (Conditional) Creating Flex Filter Assets

Create a flex filter asset whose functionality is defined by one of the default WebCenter Sites filter classes.

Before you can create flex filter assets, the flex attributes that you plan to use as the input and output attributes must be created. Create the appropriate flex attributes if they don't exist yet. Note the following requirements:

- For flex filters that use the Document Transformation filter type, the input and output attributes must be of type `blob`.

- For any flex filter, the input attribute, output attribute, and flex filter asset must all belong to the same flex family.

For more information about flex filter classes, and detailed instructions on creating flex filter assets, see Creating Flex Filters.

To create a flex filter asset:

1. In the menu bar, click **New**.

2. In the list of asset types, select the type of filter you want to create. For example, **New Media Filter**. To be able to select assignees for the workflow associated with this asset, choose at least one user from each role and click **Set Assignees**.

   The Media Filter form opens.

Figure 6-5 Media Filter Dialog

3. In the New form, do the following:

- In the **Name** field, enter a unique name for this filter asset.

- In the **Description** field, enter a brief description summarizing the filter's function.

- In the **Filters** field, select the filter class that will define the functionality of the flex filter asset you are creating. By default, the filter options are **Doc-Type**, **Document Transformation**, **FieldCopier**, and **ThumbnailCreator**.

  > **Note:**
  >
  > Custom filter classes that may have been created for your system appear in this list. For information on creating a custom flex filter class, see Defining a Flex Filter Class and Creating a Flex Filter Asset.

- Click **Get Arguments**. In the **Arguments** field, specify the input and output arguments for the flex filter asset. Click **Add** to add the arguments to the filter.

4. Click **Save**.

6.6.7 Creating Parent Definition Assets

To create a parent definition asset:

1. In the menu bar, click **New**.

2. Select the name of your product definition asset from the list of shortcuts. For example, **New Product Parent Definition**. To be able to select assignees for the workflow associated with this asset, choose at least one user from each role and click **Set Assignees**.

   The Product Parent Definition form opens.

Figure 6-6   Product Parent Definition Dialog

3. In the Product Parent Definition form, fill in the fields as follows:

    a. For **Name**, enter a name of up to 64 characters.

    b. For **Description**, enter a short descriptive phrase that describes the parent definition.

    c. For **Parent Select Style**, determine how flex parents that use this definition will be selected on the parent asset forms. Do one of the following:

        •   If the number of parents of this type will be small, choose **Select Boxes**. Then, all the parents of this type will be displayed as options in a drop-down list on the flex asset forms.

- If the number of parents of this type will be large, choose **Pick From Tree**. Then, when you select a parent of this type on the flex asset form, you select it from the tree on the tab that displays your catalog data.

    d. In the **Product Parent Definitions** section, select a parent definition from the **Available** list and then click one of the buttons described in the following table:

Table 6-1 Buttons in Parent Definition Form

Button in parent definition form	Creates a field in the New parent form that does the following:
Single Value: **Required**	Forces you to select one parent for the field.
Single Value: **Optional**	Lets you select only one parent for the field.
Multiple Value: **Required**	Forces you to select at least one parent asset for the field.
Multiple Value: **Optional**	Lets you select multiple parent assets for the field.

WebCenter Sites moves the parent definition from the **Available** list to the **Selected** list.

    e. Repeat step 3.d as many times as necessary. Remember that the corresponding New parent form will include a field for each item that you select in the **Available** list on this parent definition form.

    f. In the **Attributes** section, select the appropriate attributes from the **Available** list, and then do one of the following:

- Click **Required** to specify that the attribute is required, that is, all flex parents created with this definition must have a value for this attribute.

- Click **Optional** to specify that the attribute is optional.

Note:

To assign a flex filter asset to this parent definition in the future, you must include the input and output attributes that the flex filter uses.

To order the attributes as you want them to appear on the parent form for parents of this type, use the **Display Order** arrows to the right of the **Selected** box.

    g. In the **Filters** section, select any flex filter assets that are appropriate for this parent definition.

4. Click **Save**.

5. Repeat this procedure for each parent definition asset that you have to create.

6.6.8 Creating Flex Definition Assets

To create a flex definition asset:

1. In the menu bar, click **New**.

2. Select the name of your flex definition asset type from the list of shortcuts. For example, **New Product Definition**. To be able to select assignees for the workflow associated with this asset, choose at least one user from each role and click **Set Assignees**.

    The Product Definition form opens.

3. In the Product Definition form, fill in the fields as follows:

    a. In the **Name** field, enter a name of up to 64 characters.

    b. In the **Description** field, enter a short, descriptive phrase that describes the parent definition.

    c. In the **Product Parent Definitions** section, select a parent definition from the **Available** list and then click one of the buttons described in the following table:

    Table 6-2   Buttons in Flex Definition Form

Button in flex definition form	Creates a field in the New flex asset form that does the following:
Single Value: **Required**	Forces you to select only one parent in the field.
Single Value: **Optional**	Lets you select only one parent in the field.
Multiple Value: **Required**	Forces you to select at least one parent asset in the field.
Multiple Value: **Optional**	Lets you select multiple parent assets in the field.

    WebCenter Sites moves the parent definition from the **Available** list to the **Selected** list. For information about selecting parent definitions, see Determining Hierarchical Place.

    d. Repeat the previous step as many times as is necessary. Remember that the corresponding New Flex Asset form will include a field for each item that you select in the **Available** list on this flex definition form.

    e. In the **Attributes** section, select attributes from the **Available** list, and then do one of the following:

    •   Click **Required** to specify that the attribute is required; that is, that all flex assets created with this definition must have a value for this attribute.

    •   Click **Optional** to specify that the attribute is optional.

    Note:

    To assign a flex filter asset to this flex definition in the future, you must include the input and output attributes that the flex filter uses.

To order the attributes as you want them to appear on the New and Edit forms for flex assets created with this definition, use the **Display Order** arrows to the right of the **Selected** box.

f. In the **Filters** section, select any flex filter assets that are appropriate for this flex definition.

4. Click **Save**.

5. Repeat this procedure for each flex definition that you have to create.

6.6.9 Creating Flex Parent Assets

To create flex parent assets:

1. Log in to WebCenter Sites as an administrator and select the **Contributor** interface.

2. In the menu bar, select **Content**, then **New**, and then the *type of flex parent asset you want to create*.

   A tab opens displaying the Create view of the flex parent asset you want to create.

   Figure 6-7   Product Parent Dialog

3. In the Product Parent form, fill in the fields as follows:

   a. In the **Name** field, enter a name of up to 64 characters.

   b. In the **Product Parent Definition** field, select a parent definition from the drop-down list. The definition you select formats the next sections of the form (the sections you fill out to define this parent asset).

   c. Click **Continue**.

   The Content section of the form opens.

Figure 6-8   Content Section

4. In the Content section of the form, fill in the fields keeping in mind the following:

   • An asterisk (*) next to the field indicates that the field is required.

   • In this example, the field in which you designate a parent asset is called `FSIIProductTopLevel`. The parent that you drag and drop into this field becomes the grandparent of any flex assets you designate as children of the parent you are creating in this procedure. If this field is not required, and you do not select any parents (grandparents), the parent you are creating will be a top-level parent in the **Content** tree.

   • To determine whether the field is single or multi-valued, point to the drop zone associated with the field. A tooltip is displayed showing the type of assets this field accepts along with information about whether the field is single or multi-valued.

5. Use the form section selector to switch to the next sections of the form (**Marketing** and **Metadata** sections). An asterisk (*) next to the field indicates that it is a required field.

   The fields displayed in these sections are based on the parent definition you chose for this parent. The values that you enter into these fields are inherited by any flex assets that have this parent asset as their parents.

6. In the asset's toolbar, click the **Save** icon.

   WebCenter Sites writes the new parent to the database. All the information other than the attribute values are written to the `FlexParent`, `FlexParent _AMap`, and `FlexParent_Extension` tables, where `FlexParent` represents the internal name of your flex parents. The attribute values are written to the `FlexParent _Mungo` table.

6.6.10 Creating and Assigning Asset Type Icons (Contributor Interface Only)

When a user performs a search in the Contributor interface and displays the results of the search in the Thumbnail view, each asset in the search results list is represented by

a thumbnail image. By default, the name, asset type, modification date, and locale of the assets in the search results list are displayed below the thumbnail image representing the asset or the asset's type. Images can either be assigned per asset or per asset type. The focus of this section is to assign a thumbnail image for each asset type.

You must create and assign images that uniquely identify the members of your new flex family.

1. For each flex family member, do the following:

    a. Create two image files of type JPG, one to be displayed in the docked search results list, and the other to be displayed in the undocked search results list. The standard size of the image that will be displayed in the docked search results list is 96x96 pixels. The standard size of the image that will be displayed in the undocked search results list is 170x170 pixels.

    b. Name the image files as follows:

        •   To name the image that will be displayed for the asset type in the docked search results list, use the syntax `assetType.jpg`.

        •   To name the image that will be displayed for the asset type in the undocked search results list, use the syntax `assetType_large.jpg`.

    Note:

    You must save the images as JPGs. The file name determines the asset type for which the icon will be displayed. The name is case-sensitive.

    ---

    c. Place the image files in the appropriate directory:

        `<cs_app_dir>/images/search`

        where `<cs_app_dir>` is the directory of the deployed WebCenter Sites application on your application server.

2. Restart your application server for the icons to appear in the Thumbnail view of the search results list in the Contributor interface.

6.6.11 Coding Templates for the Flex Assets

Creating your flex asset definitions and coding the templates for the flex assets that use those definitions is an iterative process. Although you have to create definitions and flex assets before you can create templates for your flex assets, it is likely that you will discover areas that need refinement in your data design only after you have coded a template and tested the code.

For information about coding elements for your templates, see Creating Template, CSElement, and SiteEntry Assets and Coding Elements for Templates and CSElements.

6.6.12 Testing Your Design (Creating Test Flex Assets)

To thoroughly test your design, you must examine where flex assets appear on the tree, what their forms look like, how long it takes to load their forms, and so on. Create some flex assets and examine them.

For information about creating assets, see *Oracle Fusion Middleware Using Oracle WebCenter Sites*.

6.6.13 (Conditional) Creating Flex Asset Associations

In most cases, you should use a flex asset's attributes to form associations. In the rare case that your associations must work across flex definitions, create associations between flex assets.

To create associations:

1. In the **General Admin** tree, expand the **Admin** node, and then expand the **Asset Types** node.

2. Expand the node for the asset type you want to create an association for.

3. Expand the node for **Asset Associations**.

4. Click **Add New**.

    The Add New Association form opens.

    Figure 6-9   Add New Association Dialog

5. In the Add New Association form, fill in the fields as follows:

    *   In the **Name** field, enter a name for the association.

    *   In the **Description** field, enter a description of the association.

- In the **Child Asset** field, select a child asset to associate with this asset.

- In the **Subtypes** field, select one or more subtypes for this association.

- In the **Mirror Dependency Type** field, select a dependency type for the associated flex asset.

- In the **Type of Association** field, select one of the following:

    - Single valued - only one value allowed per asset.

    - Multivalued - multiple values allowed per asset.

6. Click **Add New Association** to associate the flex asset types.

6.6.14 Moving Asset Types to Other Systems

After you finish creating your flex family, creating the data structure assets (including attribute editors), and coding templates for the flex asset type, system administrators can configure the asset types for the management system. They enable revision tracking where appropriate, create workflow processes, create start menu shortcuts, and so on.

Move your flex family, data structure assets, and coding templates to the management and delivery systems so system administrators can configure them. For information about moving your asset types to the management and delivery systems, see *Oracle Fusion Middleware Administering Oracle WebCenter Sites*.

6.7 What You May Need to Know About Editing Flex Attributes, Parents, and Definitions

Some editing tips that you can use when editing flex asset types to prevent schema changes that lead to data loss.

See these topics:

- What You May Need to Know About Editing Attributes

- What You May Need to Know About Editing Parent Definitions and Flex Definitions

- What You May Need to Know About Editing Parents and Flex Assets

6.7.1 What You May Need to Know About Editing Attributes

Note the following when editing a flex attribute:

- You can change the **Name** without causing a schema change. However, to import flex assets into your WebCenter Sites database using XMLPost, you must edit your XMLPost files if you change the name of an attribute.

- You can change the **Description** without causing data loss.

- If you change the data type in the **Value Type** field, you lose all data associated with the attribute in the _Mungo table(s) that use this attribute type.

- If the attribute's data type is asset and you change the asset type, all existing data for the attribute is invalid.

- If you change the **Folder** field for a `blob` attribute, WebCenter Sites will no longer be able to find any existing data for that attribute. If you must change this value, move the file system to match the new value that you set.

- You can change the **Number of Values** from single to multiple without causing data loss or complications.

- If you change the **Number of Values** from multiple to single, WebCenter Sites cannot determine which of the values in any existing rows are the values to keep.

- You can change the **Search Engine and ISO Character Set** without causing data loss.

6.7.2 What You May Need to Know About Editing Parent Definitions and Flex Definitions

Note the following when editing a parent definition or a flex definition:

- You can change the **Name** without causing a schema change. However, to import flex assets into your WebCenter Sites database using XMLPost, you must edit your XMLPost files if you change the name of a parent definition.

- You can change the **Description** and the **Parent Select Style** fields without causing data loss.

- If you change the parent selections:

  - Adding parents is allowed.

  - Removing parents can cause assets to no longer have valid data.

  - Changing parents from optional to required can cause problems because parents or flex assets who do not have one of the newly required parents are no longer valid.

  - Changing parents from required to optional is allowed.

  - Changing parents from single value to multiple value is allowed.

  - Changing parents from multiple value to single value causes unpredictable results because WebCenter Sites cannot determine which of the previously acceptable multiple values is the one to keep and which ones to remove.

- If you change the attribute selections:

  - Adding optional attributes is allowed.

  - Adding required attributes causes existing parents or flex assets without them to be invalid.

  - Removing attributes causes existing parents or flex assets with such an attribute value to be invalid.

6.7.3 What You May Need to Know About Editing Parents and Flex Assets

Note the following when editing a flex or parent asset:

- You can change the Name without causing a schema change. However, to import flex assets into your WebCenter Sites database using XMLPost, you must edit your XMLPost files if you change the name of a parent definition.

- You can change the Description without causing data loss.

- If you change parents, WebCenter Sites corrects all the inherited attribute values.

- You cannot change the definition that you used to create the parent or flex asset.

- Changing the value of an attribute is allowed. If you change the value of an attribute for a parent, WebCenter Sites corrects that attribute for all the assets that inherited it from this parent. Changing the attribute value for a flex asset is allowed.

6.8 Using Product Sets

When you're managing an online catalog with WebCenter Sites, using the product set feature with product assets can help you group products that are identical but are packaged and sold differently.

See these topics:

- About Using Product Sets

- Creating a Product Set

6.8.1 About Using Product Sets

A book is the same book whether it is the paperback version or the hard-cover version. And a soft drink is the same soft drink whether it is sold in individual cans, as a six-pack, in a 2-liter bottle, or a case. Product sets allow you to group products like these together so that they can be displayed together (in the same form) on the management system, yet remain individual saleable units, identified as such by their SKUs.

The model for the product set feature is as follows:

- The product set is a product parent that takes on the characteristics of a product asset. The product set (parent) has all of the attributes that define the core product.

- The product assets are SKUs. That is, they have only those attributes that describe the packaging or are the unique identifiers for members of the set: the SKU, the bottle size, and so on.

- The product set (parent) has an attribute that marks it as a product set and the value of this attribute is unique among all the product sets. This attribute is called `GAProductSet` and is a reserved name. The products in the set inherit this attribute and, by this inheritance, are marked as members of that product set (that is, children of that product parent).

6.8.2 Creating a Product Set

To create a product set:

1. Create a product attribute named `GAProductSet`. This is a reserved name and your attribute name must match it exactly.

2. Create a new product parent definition and select the `GAProductSet` attribute.

3. Create a new product definition and designate that the parents created with the definition that you created in step 2 can be parents of products created with this product definition.

4. Create a new product parent from the definition you created in step 2.

5. Using the product definition that you created in step 3, create the products in the set and designate that the parent that you created in step 4 is their product parent.

Now, when you inspect or edit the product set (product parent), each product (SKU) in the set is listed on the Product Parent form, presenting a representation of the product set relationship.

There can be only one `GAProductSet` attribute in the WebCenter Sites database. To create product sets in multiple WebCenter Sites sites, you must share the `GAProductSet` attribute to the sites that you want to use it in.

7

Creating a Hierarchical Flex Family

Creating a flex family with a multi-level hierarchy and single-valued definitions will give you a basic understanding of the flex asset model in WebCenter Sites. On completing these steps, you'll also have a model that you might want to use to create similar hierarchies.

Topics:

- Hierarchical Organization

- Flex Family Specifications

- Creating a Sample Flex Family Using a Real-World Example

Hierarchical Organization and Flex Family Specifications describe the flex family that you will create. Creating a Sample Flex Family Using a Real-World Example is a tutorial, where you create a small flex family with generic names for its family members.

7.1 Hierarchical Organization

You're going to create a flex family with three levels: a top-level parent, a second level parent, and assets.

- A top-level parent (named Parent 1 [Level 1] in our example).

- A second-level parent (named Parent 2 [Level 2] in our example).

- Assets, at the third level. One asset is placed directly under its level 1 parent; another asset is placed directly under its level 2 parent; the third asset is placed under both the level 1 parent and the level 2 parent.

In the WebCenter Sites interface, the hierarchy looks exactly as shown in the left side of the following figure. The representation is formulaic. The right side of this graphic shows how it translates to a real-world model, represented by the avisports sample site.

Figure 7-1   Formulaic Data Model vs. Real World Data Model
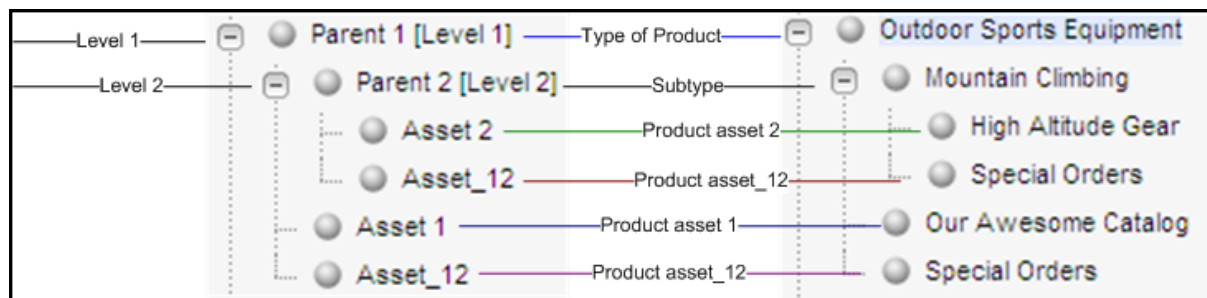
On the sample site (Right side of the above figure):

- Parent 1 [Level 1] is named Outdoor Sports Equipment.

- Parent 2 [Level 2] is named Mountain Climbing, a subtype of Outdoor Sports Equipment.

- Asset 1 is named Our Awesome Catalog, an asset of the type Outdoor Sports Equipment.

- Asset_12 is named Special Offers. It displays under both Outdoor Sports Equipment and Mountain Climbing.

7.2 Flex Family Specifications

Flex family members that you'll create in your flex family.

The following table lists the flex family members that you will create:

Table 7-1   Flex Family Members

Flex Family Member	Name	Instances	Based on Parent Definition	Based on Flex Definition
Flex Attribute Type	My Attribute	Attribute_1[1] Attribute_2	n/a	n/a
Flex Parent Definition Type	My Parent Definition	Level 1 Def Level 2 Def	n/a Level 1 Def	n/a
Flex Definition Type	My Flex Definition	Flex Def 1 Flex Def 2 Flex Def_12	Level 1 Def Level 2 Def Level 1 Def *and* Level 2 Def	n/a
Flex Parent Type	My Parent	Parent 1 [Level 1] Parent 2 [Level 2]	Level 1 Def Level 2 Def	n/a
Flex Asset Type	My Asset	Asset 1 Asset 2 Asset_12	n/a	Flex Def 1 Flex Def 1 Flex Def_12
Flex Filter Type	n/a	n/a	n/a	n/a

[1] Suffixes 1, 2 and _12 refer to levels of the hierarchy (_12 denotes both levels 1 and 2). For example, Asset 1 denotes an asset that is placed under level 1. Asset_12 denotes an asset that is placed under both levels 1 and 2.

7.3 Creating a Sample Flex Family Using a Real-World Example

You'll create a small flex family whose family members you'll give generic names. This approach will help you visualize the formula for building hierarchies.

At the end of this tutorial, you will change the names of selected family members to real-world names to understand how a formulaic data model translates to a business-

related data model. You will also add more parents and assets to the hierarchy, giving them real-world names as you create them.

This section includes the following topics:

- Creating a Flex Family

- Enabling the New Flex Asset Types

- Adding a Flex Family Tab to the WebCenter Sites Tree

- Creating Parent Definition Assets

- Creating Flex Parent Assets

- Creating Flex Definition Assets

- Creating Flex Assets

- Translating the Formulaic Data Model into a Real-World Data Model

- Developing Your Real-World Model

7.3.1 Creating a Flex Family

In this step, you create a flex family by naming its required members.

To create the flex family:

1. Launch the Admin interface.

2. In the **General Admin** tree, expand the **Admin** node, then expand **Flex Family Maker**, and then double-click **Add New Family**.

3. In the Flex Family Maker form, fill in the fields exactly as shown in the following table:

Table 7-2   Flex Family Maker Form

Field Name	Enter	Comments
Flex Attribute	MyAttribute	In this step, name the database tables that WebCenter Sites creates for the flex family. The names must not contain spaces.
Flex Parent Definition	MyParentDefinition	n/a
Flex Definition	MyFlexDefinition	n/a
Flex Parent	MyParent	n/a
Flex Asset	MyAsset	n/a
Flex Filter	n/a	n/a

4. Click **Continue**.

5. In the next form, fill in the fields for each new member of the family as follows:

    a. In the **Description** field, enter the same name as in step 3, but separate the words in the name with spaces.

    The name that you enter is used throughout the WebCenter Sites interface to identify the asset type.

    b. In the **Plural** field, enter the plural form of the name used in the preceding step.

    c. Click **Add New Flex Family**.

6. Wait for WebCenter Sites to create the flex family and return the message indicating that the flex family members (asset types) were successfully installed.

7.3.2 Enabling the New Flex Asset Types

In this step, you enable the flex family members for the avisports sample site. You also create start menu items for the members, so that you can create and search for their instances in subsequent steps.

To enable the new flex asset types:

1. In the **General Admin** tree, expand the **Admin** node, and then expand the **Sites** node and complete the following steps:

    a. Expand **avisports** (the sample site where the flex family is enabled), or a site of your choice.

    b. Under that site, expand **Asset Types** and double-click **Enable**.

        i. From the list, select the asset types that you just created (MyAsset, MyAttribute, MyFlexDefinition, MyParent, MyParentDefinition).

        ii. Click **Enable Asset Types**.

    c. In the Enable Asset Types form:

        i. Ensure all **Start Menu** options are selected (so that, later, you can create and search for instances of the family members.)

        ii. Click **Enable Asset Types**.

2. Wait for WebCenter Sites to display the message indicating that the asset types have been enabled for the site.

7.3.3 Adding a Flex Family Tab to the WebCenter Sites Tree

In this step, you add a tab that tracks the creation of your flex family. You set up this tab to display selected members of the flex family as you finish creating them.

To add the tree tab:

1. In the **General Admin** tree, expand the **Admin** node, and then double-click the **Tree** node.

2. In the Tree Tabs form, scroll to the bottom and click **Add New Tree Tab**.

3. In the Add New Tree Tab form, fill in the fields as in the following table:

Table 7-3   Add New Tree Tab Form

Field Name	Enter or Select	Comments
Title	Sample Flex Family	Name of the tab.
Tooltip	Sample Flex Family	Description of the tab.
Sites	avisports (or the site you chose in Enabling the New Flex Asset Types)	Site on which to enable the flex family.
Required Roles	Any	n/a
Tab Contents	My Parent Definition My Parent My Flex Definition My Asset **Note:** Click **Add Selected Items** and use the **Display Order** arrow to arrange the members in this order.	n/a

4. Click **Save**.

5. Refresh the page.

6. Click the **Sample Flex Family** tab and make sure its contents are identical to the following figure:

Figure 7-2   Sample Flex Family Tab
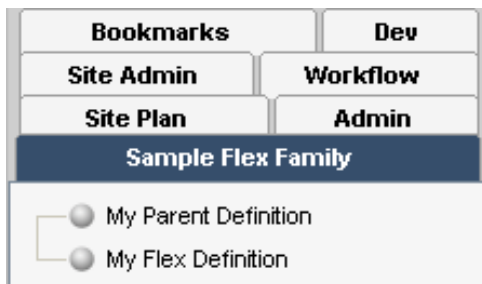
7.3.4 Creating Parent Definition Assets

In this step, you create two parent definitions. The first parent definition establishes the top level of the hierarchy, and the second parent definition establishes the second level.

To create the first parent definition asset:

1. In the menu bar, click **New**.

2. From the list of options that display, select **New My Parent Definition**.

3. In the next form:

   a.   Fill in the fields as in the table.

Table 7-4   New My Parent Definition Form

Field Name	Enter or Select	Comments
Name	Level 1 Def	This is our name for the definition of the first level of the hierarchy.
Description	Level 1 Def	N/A
Parent Select Style	Select Box	N/A
My Parent Definitions	n/a	No parent definitions are selected (or available) in this field. Therefore, this parent definition establishes the first level of the hierarchy.

    b.   Click the **Save icon**.

To create the second parent definition asset:

1.   In the menu bar, click **New**.

2.   From the list of options that display, select **New My Parent Definition**.

3.   In the next form:

    a.   Fill in the fields as in the table.

Table 7-5   New My Parent Definition Form

Field Name	Enter or Select	Comments
Name	Level 2 Def	This is our name for the definition of the second level of the hierarchy.
Description	Level 2 Def	N/A
Parent Select Style	Select Box	N/A
My Parent Definitions	Level 1 Def **Note**: Under Single Value, click the **Required** arrow to move your selection to the **Selected** list.	Choosing **Level 1 Def** subordinates the current parent definition to **Level 1 Def**. Chaining definitions in this manner establishes Level 2 Def as the second level. When parents are created and based on the current parent definition (Level 2 Def), they are subordinated to parents that are based on Level 1 Def.

    b.   Click the **Save icon**.

4.   Refresh the page.

5. Click the **Sample Flex Family** tab and expand its contents to make sure they are identical to this figure:

Figure 7-3   Sample Flex Family Tab with Parent Definition
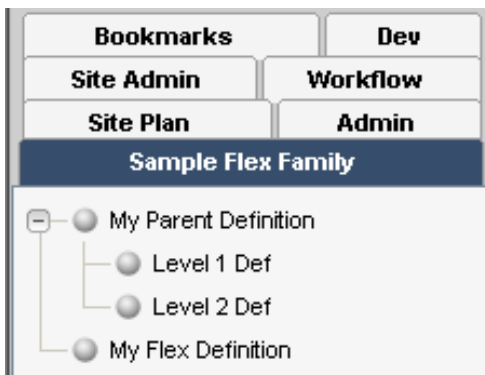
7.3.5 Creating Flex Parent Assets

In this step, you create two flex parent assets, and base them on the flex parent definitions that you created in the previous step. The first parent asset occupies the top level of the hierarchy. The second parent asset occupies the second level of the hierarchy.

To create the top-level parent of the hierarchy:

1. Switch to the Oracle WebCenter Sites: Contributor interface by clicking the **Contributor** icon on the top.

2. From the **Content** menu, choose **New**, then **New My Parent**.

3. In the form that opens, fill in the fields as in the table.

Table 7-6   New My Parent Form

Field Name	Enter or Select	Comments
Name	Parent 1 [Level 1]	This is our name for a level 1 parent in the hierarchy (a generic name simply to help you identify the level).
		Note: At the end of this tutorial, you will change the name to a business-specific name (Outdoor Sports Equipment, in our example, which describes the inventory of a company dealing with sports gear.)
My Parent Definition	Level 1 Def	Choosing **Level 1 Def** places the parent you are creating at the top level of the hierarchy.

4. On top of the form, click the **Save** icon.

To create the second-level parent of the hierarchy:

1. From the **Content** menu, choose **New**, then **New My Parent**.

2. In the form that opens, fill in the fields as in the table.

Table 7-7   New My Parent Form

Field Name	Enter or Select	Comments
Name	Parent 2 [Level 2]	This is our name for a level 2 parent in the hierarchy (a generic name to help you identify the level).
		Note: At the end of this tutorial, you will change the name to a business-specific name, Mountain Climbing in our example (an appropriate name given that Parent 1 [Level 1] is Sports Equipment).
My Parent Definition	Level 2 Def	Selecting **Level 2 Def** places the parent you are creating at the second level of the hierarchy.
Level 1 Def	Parent 1 [Level 1] is selected by default.	N/A

3. Click the **Save** icon.

4. On the **Content Tree**, expand **Sample Flex Family** to make sure its contents are identical to the figure.

Figure 7-4   Sample Flex Family Tab Expanded on the Content Tree
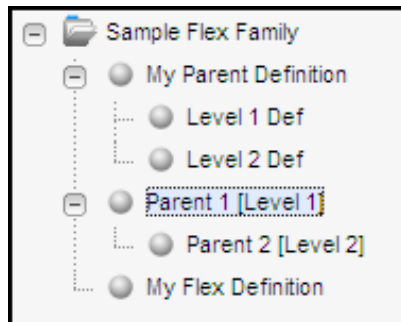
Note:

Before going to the next step, review the following figure that summarizes how the objects that you created, parent definitions and parents based on the definitions, relate to each other.

Figure 7-5   Parents and Parent Definitions
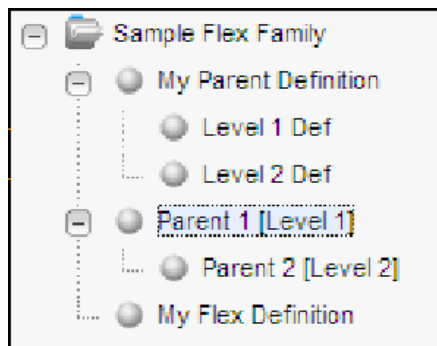
7.3.6 Creating Flex Definition Assets

In this step, you create three flex definition assets:

- The first flex definition asset is used to place assets under Parent 1 [Level 1].

- The second flex definition asset is used to place assets under Parent 2 [Level 2].

- The third flex definition asset is used to place the asset under both levels of the hierarchy.

To create the first flex definition asset:

1. Switch to the Admin interface by clicking the **Admin** icon, located in the applications bar.

2. In the menu bar, click **New**.

3. From the list of options that display, select **New My Flex Definition**.

4. In the form that opens, fill in the fields as in the table.

Table 7-8   New My Flex Definition Form

Field Name	Enter or Select	Comments
Name	Flex Def 1	N/A
My Parent Definitions	Level 1 Def **Note:** Under Single Value, click the **Required** arrow.	Choosing **Level 1 Def** and **Single Value** means that when you use the current flex definition to create flex assets, the assets can be placed under *only one* parent that use Level 1 Def as its parent definition. In our example, the asset is placed under Parent 1 [Level 1]. **Note**: Selecting a Multiple Values option allows you to place the asset under *any* and *all* parents that use Level 1 Def as their parent definition.

5. Click the **Save** icon.

To create the second flex definition asset:

1. From the button bar, click **New**.

2. From the list of options that display, select **New My Flex Definition**.

3. In the form that opens, fill in the fields as in the table.

Table 7-9   New My Flex Definition Form

Field Name	Enter or Select	Comments
Name	Flex Def 2	N/A

Table 7-9   (Cont.) New My Flex Definition Form

Field Name	Enter or Select	Comments
My Parent Definitions	Level 2 Def **Note**: Under Single Value, click the **Required** arrow.	Choosing **Level 2 Def** and **Single Value** means that when you use the current flex definition to create flex assets, the assets can be placed under *only one* parent that uses Level 2 Def as its parent definition. In our example, the asset is placed under Parent 2 [Level 2]. **Note**: Selecting a Multiple Values option allows you to place the asset under *any* and *all* parents that use Level 2 Def as their parent definition.

4. Click the **Save** icon.

To create the third flex definition asset:

1. From the button bar, click **New**.

2. From the list of options that display, select **New My Flex Definition**.

3. In the form that opens, fill in the fields as in the table.

Table 7-10   New My Flex Definition Form

Field Name	Enter or Select	Comments
Name	Flex Def_12	N/A
Parent Definitions	Level 1 Def Level 2 Def **Note:** Under Single Value, click the **Required** arrow.	Choosing **Level 1 Def** *and* **Level 2 Def** and **Single Value** means that when you use the current flex definition to create flex assets, the assets are placed under *only one* parent that uses Level 1 Def *and* under *only one* parent that uses Level 2 Def as parent definitions. In our example, the assets are placed under Parent 1 [Level 1] *and* Parent 2 [Level 2]).
Parent Definitions (*continued*)	N/A	**Note**: Selecting a Multiple Values option allows you to place the asset under *any* and *all* parents that use Level 1 Def and Level 2 Def as their parent definitions.

4. Click the **Save** icon.

5. Refresh the page.

6. Click the **Sample Flex Family** tab and expand its contents to make sure they are identical to the following figure.

Figure 7-6   Sample Flex Family Tab
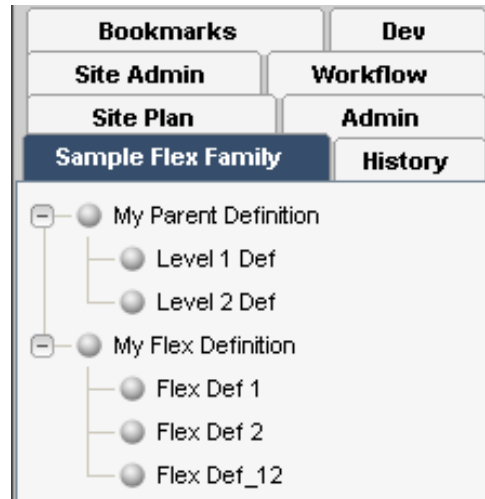
Note:

Before going to the next step, review the next figure, which depicts the Sample Flex Family structure as it displays in the Contributor application. This figure shows how the objects that you created, flex definitions, relate to the parent definitions they are based upon. Flex Definition 1 is based on Level 1 Parent Definition. When used to create assets, this flex definition places the assets under Parent 1 [Level 1]. Flex Definition 2 is based on Level 2 Parent Definition. When used to create assets, this flex definition places the assets under Parent 2 [Level 2]. Flex Definition 12 is based on Level 1 and Level 2 Parent Definitions. When used to create assets, this flex definition places the assets under both Parent 1 [Level 1] and Parent 2 [Level 2].

Figure 7-7   Flex Definitions and Assets
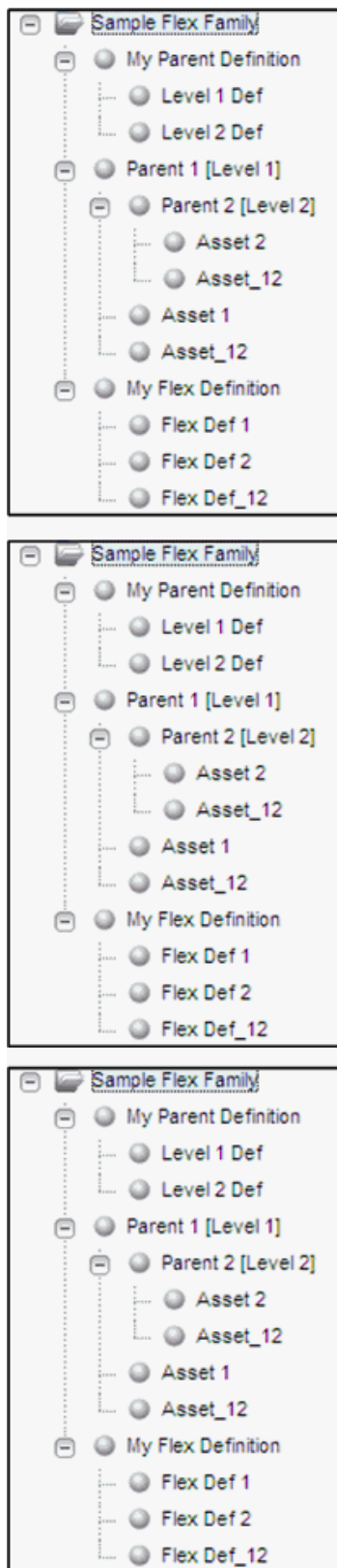
7.3.7 Creating Flex Assets

In this step, you complete the flex family by adding the third level of the hierarchy; the flex assets. You create three assets:

- The first asset you place under Parent 1 [Level 1].

- The second asset you place under Parent 2 [Level 2].

- The third asset you place under both Parent 1 [Level 1] and Parent 2 [Level 2].

To create the first flex asset:

1. Switch to the Contributor interface by clicking the **Contributor** icon on the top.

2. From the **Content** menu, choose **New**, then **New My Asset**.

3. In the form that opens, fill in the fields as in the table.

Table 7-11   New My Asset Form

Field Name	Enter or Select	Comments
Name	Asset 1	N/A
My Flex Definition	Flex Def 1	Choosing Flex Def 1 places the asset you are creating under Parent 1 (Level 1].

4. Click the **Save** icon.

To create the second flex asset:

1. Switch to the Contributor interface by clicking the **Contributor** icon on the top.

2. From the **Content** menu, choose **New**, then **New My Asset**.

3. In the form that opens, fill in the fields as in the table.

Table 7-12   New My Asset Form

Field Name	Enter or Select	Comments
Name	Asset 2	N/A
My Flex Definition	Flex Def 2	Choosing Flex Def 2 places the asset you are creating under Parent [Level 2].

4. In the next form, click **Save**.

To create the third flex asset:

1. Switch to the Contributor interface by clicking the **Contributor** icon on the top.

2. From the **Content** menu, choose **New**, then **New My Asset**.

3. In the form that opens, fill in the fields as in the table.

Table 7-13   New My Asset Form

Field Name	Enter or Select	Comments
Name	Asset_12	N/A
Flex Definition	Flex Def_12	Choosing Flex Def_12 places the asset you are creating under both Parent [Level 1] *and* Parent [Level 2].

4. In the next form, click the **Save** icon.

5. Refresh the page.

6. Click the **Sample Flex Family** tab and expand its contents to make sure they are identical to the following figure.

Figure 7-8   Sample Flex Family Structure
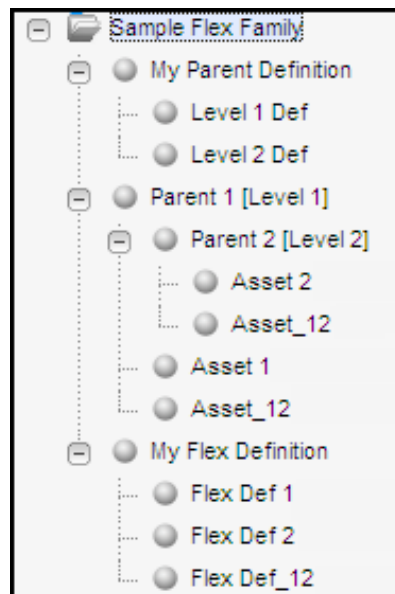
The following figure shows how the objects that you created, assets, relate to the flex definitions they are based upon. Based on the definition, each asset is placed under a relevant parent. For instance, the Flex Def 1 definition places the Asset 1 asset under Parent 1 [Level 1]. However, in case of Flex_Def_12 Definition, the Asset_12 asset is placed under both Parent 1 [Level 1] and Parent 2 [Level 2].

Figure 7-9   Flex Definitions and Assets
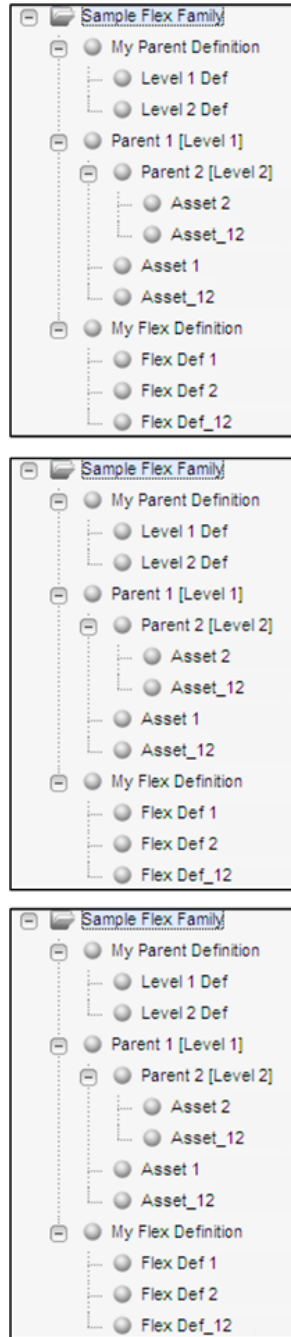
7.3.8 Translating the Formulaic Data Model into a Real-World Data Model

In this step, you rename the parent definitions, parents, and assets to translate the formulaic data model you just created into a real-world model. (In practice, instead of renaming flex family members, you would name them directly in their business context, as you create them.)

In our example, the real-world model describes a business that deals with sports gear. The flex parents and assets, after you finish renaming them, are listed in your **Content Tree** tab as shown in the following figure.

Figure 7-10   Content Tree Tab
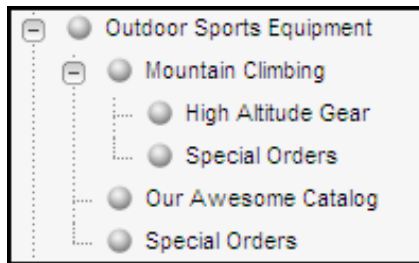
To create the real-world model:

1. Rename the parent definitions as follows:

    a. Open the Admin interface.

    b. In the Sample Flex Family Tree, right-click **Level 1 Def** and choose **Edit** from the context menu.

    c. Replace the parent's name with **Level 1 Def (Type of Sports Equipment)**, and click the **Save** icon.

    d. In the same manner, replace the name of **Level 2 Def** with **Level 2 Def (Sport)**.

2. Rename the parents as follows:

    a. Switch to the Contributor interface.

    b. Click **Content Tree** to display its contents.

    c. In the Sample Flex Family Tree, right-click **Parent 1 [Level 1]**, and choose **Edit** from the context menu.

    d. Replace the parent's name with **Outdoor Sports Equipment**, and click the **Save** icon.

    e. In the same manner, replace the name of **Parent 2 [Level 2]** with **Mountain Climbing**. The Sample Flex Family Tree should look like the following figure.

    Figure 7-11   Sample Flex Family Tree Node Expanded
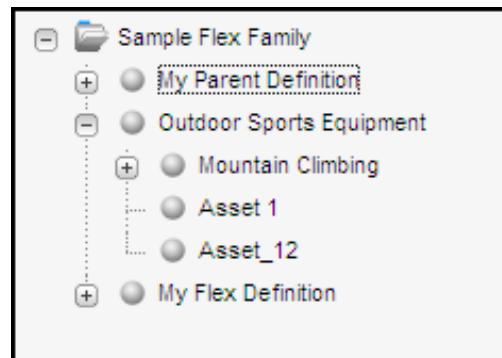
3. Rename the assets as follows:

    a. In the Sample Flex Family Tree, expand **Outdoor Sports Equipment**.

b. Right-click **Asset 1** and choose **Edit**.

c. Replace the asset's name with **Our Awesome Catalog**, and then click the **Save** icon.

d. In the same manner, expand **Mountain Climbing** and replace the name of **Asset 2** with **High Altitude Gear**.

e. Replace the name of **Asset_12** with **Special Orders**.

The assets you renamed should look like the following figure.

Figure 7-12   Sample Flex Family Tree Node Expanded

7.3.9 Developing Your Real-World Model

In this step, you develop your data model by creating a new parent and its asset, giving each a real-world name. You do the following:

• Create a second level-2 parent and name it **White Watering**.

• Create the parent's asset (a catalog) and name it **Rafts, Canoes, and Kayaks**. Expand White Watering to display its assets, as shown in this figure:

Figure 7-13   Sample Flex Family Tree Node Expanded
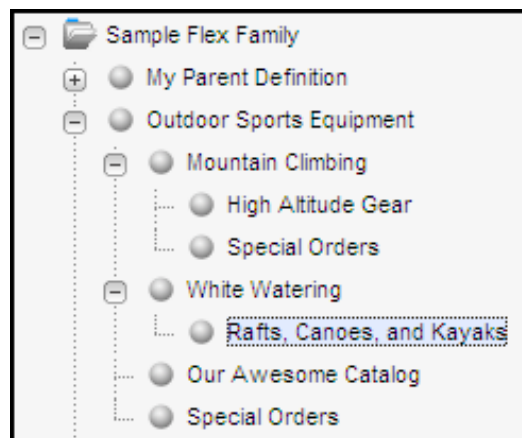
To create the level-2 parent:

1. Switch to the Contributor interface by clicking the **Contributor** icon on the application bar.

2. From the **Content** menu, choose **New**, then **New My Parent**.

3. In the form that opens, fill in the fields as in the table:

Table 7-14   New My Parent Form

Field Name	Value
Name	White Watering
My Parent Definition	Level 2 Def (Sports)

4. Click the **Save** icon.

To create the parent's asset:

1. Switch to the Contributor interface by clicking the **Contributor** icon on the application bar.

2. From the **Content** menu, choose **New**, and then **New My Asset**.

3. In the form that opens, fill in the fields as in the table:

Table 7-15   New My Asset Form

Field Name	Value
Name	Rafts, Canoes, and Kayaks
My Flex Definition	Flex Def 2

4. From the **Level 2 Def** drop-down, choose **White Watering**.

5. Click the **Save** icon.

6. Refresh the page and display the **Sample Flex Family** tree tab.

   Its content should be identical to this figure:

Figure 7-14   Sample Flex Family Tree Node Expanded

8
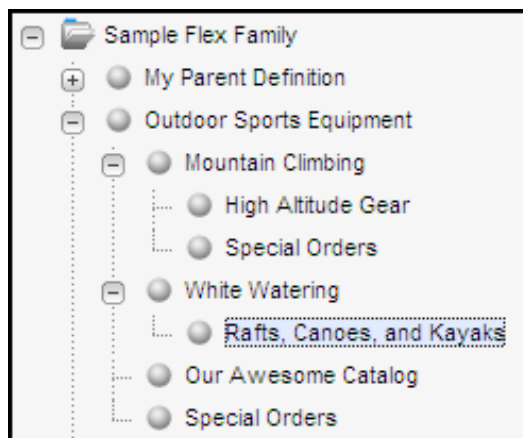
Creating Flex Filters

The flex assets you create can work appropriately when you associate flex filters with them. The flex filters carry out the assigned tasks, while flex classes implement the flex filters' functionality. You can associate multiple flex filters with a flex asset and define the order of processing, too.

Topics:

- About Flex Filter Classes and Assets
- Defining a Flex Filter Class and Creating a Flex Filter Asset
- Document Transformation Flex Filter
- SampleFlexFilter.java

8.1 About Flex Filter Classes and Assets

Flex classes implement the functionality of flex filter assets so that the filters can process the tasks you've designed them for.

See these topics:

- Flex Filter Classes
- Flex Filter Assets

8.1.1 Flex Filter Classes

Flex filter classes implement the functionality of flex filter assets. These classes are listed in the `Filters` table in the WebCenter Sites database. When you create a flex filter asset, you select a flex filter class for it.

WebCenter Sites delivers the following flex filter classes:

- **Doc-Type**: Extracts components from an asset containing one or more MIME file types and maps each file type to an individually named attribute.

- **Thumbnail Creator**: Converts an image into a thumbnail.

- **Field Copier**: Copies the contents of a system-defined attribute into a user-defined attribute.

- **Document Transformation**: Converts a document from one file type into another by invoking a registered transformation engine (an engine that is specified in the `SystemTransforms` table). The transformation engine functions as a wrapper that forwards calls to a document transformer, which then performs document conversion.

You can create a custom flex filter class by defining it in the WebCenter Sites `Filters` database table. For instructions on creating a custom flex filter class, see Defining a Custom Flex Filter Class.

Doc-Type Filter Class

A Doc-Type filter takes a document and extracts the MIME-type data associated with the file into its individual components. For example, an uploaded file containing text and a GIF photo would be filtered into two generated attributes, one containing TXT formatted data and one containing GIF formatted data.

The Doc-Type class is defined by the following arguments:

- **Attribute to Hold Derived File Name**: (Optional) Enter the flex attribute that stores the output file name.

- **Attribute to Hold Derived File Type**: The file extension is stored in this attribute.

- **Attribute to Hold Derived MIME Type**: (Optional) MIME files can have several types such as plain text, attachment, media file, and so on.

- **Input Attribute Name**: The attribute name stored by WebCenter Sites corresponds to the uploaded file name.

Thumbnail Creator Filter Class

With the thumbnail creator filter, a content provider can upload an original graphic and WebCenter Sites can create a new thumbnail sized GIF graphic file.

The Thumbnail Creator class is defined by the following arguments:

- **Input Attribute Name**: The name of the uploaded file.

- Display values for the large version of the thumbnail graphic: **Output Attribute for Main Height**, **Output Attribute for Main Width**, and **Enter Maximum Pixel Size**.

- Attributes that define the thumbnail to be created: **Output Attribute Name**, **Output Attribute for Thumb Height**, **Output Attribute for Image Aspect**, and **Output Attribute for Thumb Width**.

Field Copier Filter Class

The field copier filter copies the contents of a system-defined attribute into a user defined flex attribute.

The Field Copier class is defined by the following arguments:

- **Name**: The name of the system-defined attribute you want to copy.

- **Value**: The name of the flex attribute into which you are copying the system-defined attribute's value.

Figure 10-3 illustrates an advanced example of how to implement a field copier filter, using the Media flex family of the FirstSiteII sample site. The purpose of the field copier filter in this example is to categorize image assets by the names of their parent assets.

Document Transformation Filter Class

The Document Transformation filter class is defined by the following arguments:

- **Document Transformer Name**: The name of a registered transformation engine exactly as it is listed in the `SystemTransforms` table. By default, CS: Convert to Raw Text is listed in the `SystemTransforms` table. This engine is used to initiate the conversion of a binary file to a TXT format.

Note:

The following document transformer is available with WebCenter Sites:

`com.fatwire.transformer.tika.DocumentTransformerImpl`

The document transformer is coded to convert documents to raw text files when it is invoked by the **CS: Convert to Raw Text** engine. Converting to any other file type requires writing a document transformer for that file type, registering the corresponding transformation engine (unless it is registered), and registering the document transformer with WebCenter Sites. For information on implementing default and document transformation solutions, see Document Transformation Flex Filter.

- **Input Attribute Name**: The name of the flex attribute whose contents are to be converted by the flex filter. For the Document Transformation filter, the input attribute must be of type `blob` because it expects to find a file in that attribute.

- **Fail on Transform Error**: Choose whether the system will display an error message if the transformation does not complete properly.

- **Output Attribute Name**: The name of the flex attribute that stores the results of the document transformation. For the Document Transformation filter, the output attribute must be of type `blob` because it stores the results of the transformation as a file.

  The data stored in the output attribute (field) is read-only because it is derived from the data in the input attribute. This data is regenerated from the source data in the input attribute each time the asset is saved.

- **Output Document Extension**: The file extension to be assigned to the resulting file. Enter a document extension appropriate to the selected Document Transformer Name. For example, when you specify that the document transformation engine is **CS: Convert to HTML**, the document extension must be either **htm** or **html**.

8.1.2 Flex Filter Assets

Flex filter assets are defined by all of the following criteria:

- A flex filter class registered in the `Filters` table.

- The information that is passed to that filter class (through arguments). These arguments specify which data to use, any constraints on the filter's action, and where to store the results of the filtering process when the asset is saved.

When you create a flex filter asset, you select the flex filter class to be used, then enter values for the arguments that the filter class needs to perform its action.

After you create a flex filter asset, you assign it to the appropriate flex definition assets. Then, whenever content providers save a flex asset of that definition, the filter automatically performs its assigned action. See Creating a Flex Filter Asset.

8.2 Defining a Flex Filter Class and Creating a Flex Filter Asset

Your flex asset needs a flex filter to process the flex asset's task. And, your flex filter works when you define a custom flex filter class that implements the flex filter's functionality.

These are your basic steps:

1. Create the Java class that provides the implementation code for the custom flex filter class. See Implementation of a Flex Filter Class.

2. Define the new flex filter class in the `Filters` table in the WebCenter Sites database. See Defining a Custom Flex Filter Class.

3. Create a flex filter asset that is defined by the custom flex filter class:

   a. Create the attributes that will be referenced in the filter's arguments.

   b. Create the flex filter asset and assign it to a flex filter class.

   c. Add the new filter asset to a child definition.

   d. Re-save all related assets associated with the definition to which you added the filter asset. See Creating a Flex Filter Asset.

See these topics:

- Implementation of a Flex Filter Class

- Defining a Custom Flex Filter Class

- Creating a Flex Filter Asset

8.2.1 Implementation of a Flex Filter Class

To implement a custom flex filter, create a new Java class for it by extending the `AbstractFlexFilter` class. This filter class contains all default functionality required to handle filter requests. A working example of a custom flex filter class that extends the `AbstractFlexFilter` class, and whose purpose is to access a flex attribute and set a derived attribute value, is provided by the source file SampleFlexFilter.java.

A flex filter's functionality is implemented by parameters called abstraction interfaces. Since flex filters only have to know about certain aspects of the assets they are filtering, abstraction interfaces provide filters with access to only the asset information necessary for them to perform their function on a given flex asset.

The following table lists the abstraction interfaces required to implement a flex filter, and the asset information each abstraction interface passes to the filter. For information about abstraction interfaces used to implement a custom flex filter class, see the *Oracle Fusion Middleware Java API Reference for Oracle WebCenter Sites*.

Table 8-1   Abstraction Interfaces (`com.openmarket.gator.interfaces` Package)

Abstraction Interface	Description
IFilterEnvironment	Provides methods to obtain information about the environment that is supporting the filter.
IFilterableAssetInstance	Provides methods to manipulate the asset that is being filtered.
IFilterDescription	Provides methods to describe all the potential derived attributes that will be modified by the filter during its execution.
IFilterDependencies	Provides methods to log the dependencies against the asset instance to be filtered. A dependency refers to another asset by type and identifier that shares a relationship with the asset to be filtered. When a dependency is declared, it is either exact or exists.

Note:

Standard asset attributes can be obtained by the `IFilterableAssetInstance.get` method. For example, to get the standard asset description, you would add the line:

```
String description = instance.get(description);
```

When building the Java code for the new flex filter class, define a constructor with a single `FTValList` parameter (located in the `COM.FutureTense.Interfaces` package). This parameter provides a list of arguments obtained from the filter's definition in the `Filters` database table. These arguments are passed to the filter in the form of key/value pairs. If there are no predefined arguments for the filter class in the `Filters` database table, the `FTValList` is null.

8.2.1.1 AbstractFlexFilter Class Extension

The `AbstractFlexFilter` class can be extended to build your implementation of a flex filter class. This class is located in the `com.openmarket.gator.flexfilters` package. When you create the new flex filter's Java class, you can call the required methods necessary for your filter's functionality from the`AbstractFlexFilter` class Java code. This simplifies the amount of code necessary to create a custom implementation.

When a method is called, it is provided with the argument `String filterIdentifier`, which is the asset identifier for the filter. When you associate the same filter with different flex asset families, the filter identifier reflects the filter definition for the associated family. This argument is useful when you are implementing a filter that will be used by different asset families in order for the filter to know which flex family association is being used at the moment the filter is invoked.

For information about abstract methods used to implement a flex filter class, see the *Oracle Fusion Middleware Java API Reference for Oracle WebCenter Sites*.

Required Abstract Methods

The following abstract methods are required by all flex filter implementations:

```
public void filterAsset(IFilterEnvironment env,
      String filterIdentifier,
      FTValList filterArguments,
      IFilterableAssetInstance instance)
      throws AssetException;
```

These lines are the main method to process asset post processing when a new or pre-existing asset is saved. This method is not called if the edit is canceled. It does the work that represents the filter's purpose. A list of arguments (`FTValList`) is provided so the filter can obtain input or output (or both) attribute definitions, and any other information valid to the filter. The `filterArguments` list is defined when the filter is created in the WebCenter Sites interface.

```
public FTValList getLegalArguments(IFilterEnvironment env,
      String filterIdentifier)
      throws AssetException;
```

These lines are the method that is called to return a list of legal filter arguments. The WebCenter Sites interface will call this during filter creation or editing to populate the drop-down list after selecting the filter and pressing the **Get Arguments** button.

Optional Abstract Methods

The following abstract methods can be used to override those within the `AbstractFlexFilter` class. These methods are optional because the default implementations provided by the `AbstractFlexFilter` class are usually sufficient for most filters:

```
public void describeDerivedAttributes(IFilterEnvironment env,
      String filterIdentifier, FTValList filterArguments,
      String defTypeName, String parentDefTypeName,
      IFilterDescription descriptionObject) throws AssetException
```

This method describes all the potential derived attributes, group affinities, and recommendations that the filter might set. When the filter plans to output to attributes, this method must identify the attributes that will be modified. This is called whenever the flex asset is viewed, to anticipate the editing of the asset.

```
public void getDependencies(IFilterEnvironment env,String filterIdentifier,
   FTValList filterArguments, String assetTypeName, String parentTypeName,
   IFilterDependencies filterdeps) throws AssetException
```

This method is called to describe the filter's asset dependencies. Filter dependencies are set to either `exists` or `exact`.

```
public String[] getArgumentLegalValues(IFilterEnvironment env,
   String filterIdentifier, String argumentName) throws AssetException
```

This method is called to return a list of acceptable values for a specified argument. Any value is accepted if the return list is null. This method is called by the Admin interface when a filter asset is being created or edited to validate the argument values that are specified for the filter asset.

8.2.2 Defining a Custom Flex Filter Class

Define a custom flex filter class in the `Filters` table in the WebCenter Sites database. In the following procedure, the flex filter class is named `CustomFilter`.

To define a custom Flex Filter class:

1. Copy the `.jar` or `class` file containing the implementation code for the custom flex filter class into the directory that holds the WebCenter Sites product `jars`:

   > **Note:**
   >
   > For information about creating a Java class that provides the implementation code for your custom flex filter class, see Implementation of a Flex Filter Class.

   - For WebLogic: `app-server-install-dir/bea/path-to-domain/` `domain-name/applications/WEB-INF/lib`

   - For WebSphere: `WebSphere-Installation-Directory/` `InstalledApps/WEB-INF/lib`

2. Open Oracle WebCenter Sites Explorer and add a row to the `Filters` table for the new filter class:

   a. In the tree, expand the **Tables** node, and then select the **Filters** table.

   b. Select **File**, then **New**, and then **Record**.

   c. Define the filter in the database by filling in the following columns:

      - `name`: Enter the name of the filter as it will be displayed in the WebCenter Sites interfaces.

      - `description`: (Optional) Enter a short summary about the purpose of the filter class.

      - `classname`: Enter the exact classname of the filter class' implementation (for example, `com.fatwire.firstsite.filter.SampleFlexFilter`). This name must be available for loading in the WebCenter Sites classpath.

      - `args`: (Optional) Enter the input and output arguments for the filter. Argument key/value pairs are delimited by an ampersand (&) character. (for example, `arg1=argument1&arg2=argument2`). These arguments are passed to the filter constructor and their use is left up to the filter's developer.

   > **Note:**
   >
   > If you do not define the flex filter class' input and output arguments in the `Filters` table, you can define the arguments in the flex filter class' code. See Implementation of a Flex Filter Class.

   d. Select **File** and then **Save**.

Your new filter entry looks similar to this figure:

Figure 8-1   Sample Filter Entry

name	description	classname	args
• CustomFilter	Sample Flex Filter	com.fatwire.firstsite.filter.SampleFlexFilter	Input custom string=argument1&Output custom string=argument2
• DocType	Extracts Mime Type from Document	com.fatwire.firstsite.filter.DocType	
• Document Transformation	Use a registered document transformation engine	com.openmarket.gator.filters.DocTransformation	
• FieldCopier	Copies Base Fields to Attributes	com.fatwire.firstsite.filter.FieldCopier	
• ThumbnailCreator	Creates Thumbnail	com.fatwire.firstsite.filter.Imaging	

The filter class is now displayed as an option in the **Filter** drop-down list in the New and Edit forms of filter assets.

8.2.3 Creating a Flex Filter Asset

Before you can create a filter asset, the flex attributes that you want to use as the input and output attributes must exist. To ensure that the values of preexisting attributes are not overwritten, create attributes to be referenced by the arguments for your custom flex filter. For this example, create two `Media` attributes of type `string`; `FSII_CustomInput` to be used as the input attribute and `FSII_CustomOutput` to be used as the output attribute.

Create a flex filter asset named `FSII_CustomFlexFilter` for the `Media` flex family of the FirstSiteII sample site.

To Create a Flex Filter Asset:

1. Log in to the Admin interface as a general administrator, and select the site for which you want to create a flex filter asset (FirstSiteII sample site in this example).

2. Create the attributes (if not defined) that will be referenced by the filter's input and output arguments. Note the following requirements:

   - For flex filters that use the Document Transformation filter class, the input and output attributes must be of type `blob`.

   - For any flex filter, the input attribute, output attribute, and flex filter must all belong to the same flex family.

3. Create a flex filter asset for the associated flex family (Media flex family in this example):

   a. From the start menu items, click **New**.

   b. In the list of asset types, select the type of filter you want to create. In this example, **New Media Filter**.

      The Media Filter form opens.

   c. Fill in the following fields:

      i. In the **Name** field, enter a unique name for this filter (in this example, use `FSII_CustomFlexFilter`).

      ii. In the **Description** field, enter a brief description summarizing the filter's function.

      iii. From the **Filter** drop-down list, select the filter definition that matches the name you assigned to the custom filter (in this example, select **CustomFilter**). Then click **Get Arguments**.

iv. In the **Arguments** field, specify the input and output arguments for the flex filter asset. Click **Add** to add the argument(s) to the filter.

d. Click **Save**.

This figure shows the saved filter:

Figure 8-2 New Filter
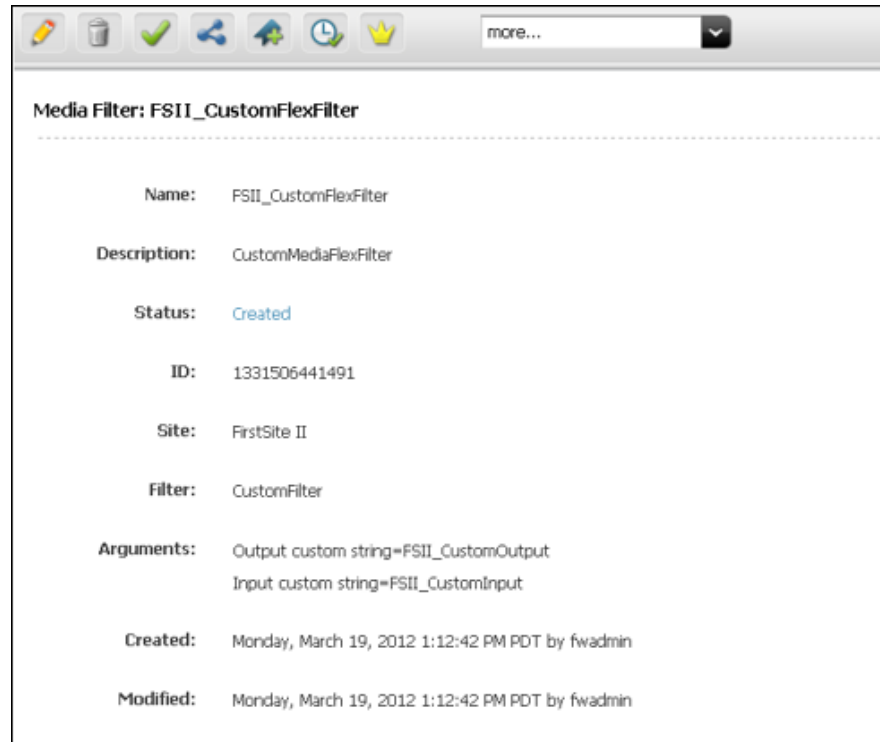
4. Find a child definition to which you want to add the new filter asset (in this example. For example, add the filter to the `Media` child definition `FSII_Image`).

a. From the start menu items, click **Search**.

b. In the list of asset types, select a type of asset definition to which you want to add the filter (**Find Media Definition** in this example).

c. In the **Search** field, enter the name of the definition to which you want to add the filter (**FSII_Image** in this example).

d. Click **Search**.

e. In the list of search results, navigate to a definition and click its **Edit** icon (**FSII_Image** in this example).

The Edit form of the definition opens.

5. In the Edit form, add the filter and input argument to the definition:

a. For **Attributes**, highlight the input attribute in the **Available** list and move it to the **Selected** list (**FSII_CustomInput** in this example).

b. For **Filters**, highlight a flex filter in the **Available** list and move it to the **Selected** list (**FSII_CustomFlexFilter** in this example).

> **Note:**
>
> Add the new filter after any other filters that will create or modify attributes which the filter you are adding depends on or shares in common.

    c. Click **Save**. This figure shows a saved filter.

Figure 8-3   New Filter
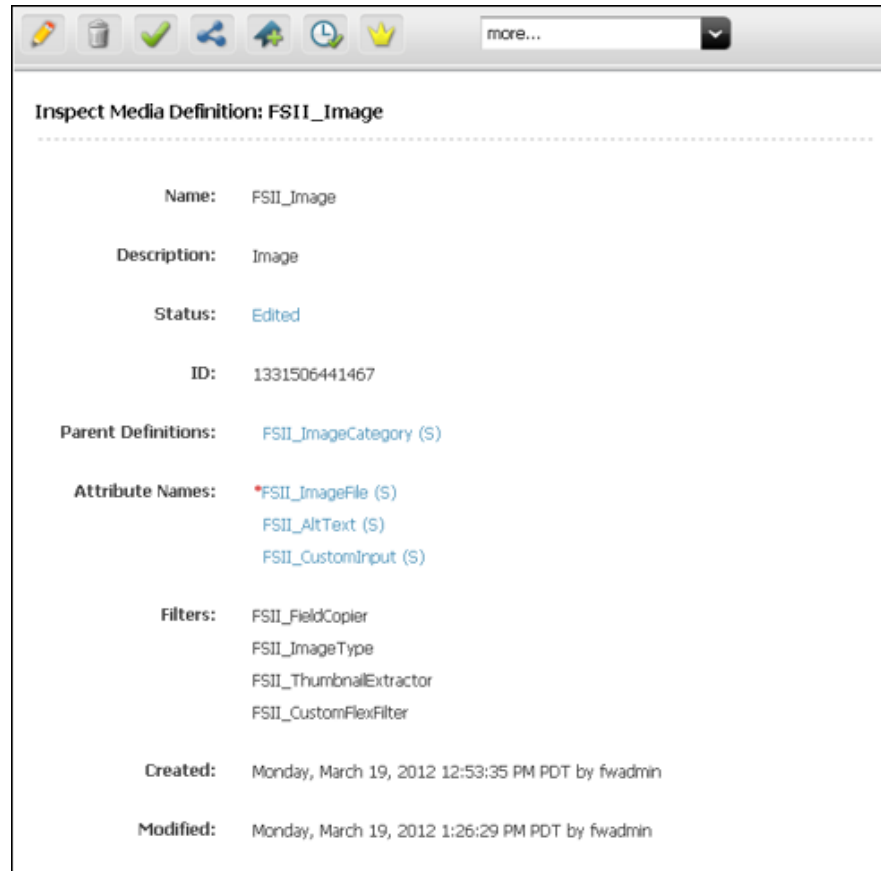
6. Find and re-save all preexisting assets associated with the definition to which you added the filter. This enables the filter to populate the output attribute (**FSII_CustomOutput**) with the derived value from the input attribute (**FSII_CustomInput**). For example:

    a. In the applications bar, click the **Contributor** icon to switch to the Oracle WebCenter Sites: Contributor interface.

    b. In the **Search** field, click the down-arrow. In the **Search Type** menu, choose the type of asset associated with the definition you modified in step 5 (**Find Media** in this example). Then click the **magnifying glass** icon.

    A **Search** tab opens displaying the results of your search.

    c. Re-save the asset. In the asset toolbar, click the **Save** icon.

    d. Inspect the asset. In the asset toolbar, click the **Inspect** icon.

    This image is the result of editing a flex asset that invokes several filters, including the filter you created in this section. By default, Media assets call

the `FSII_FieldCopier`, `FSII_ImageType`, and
`FSII_ThumbnailExtractor` filters. In this example, the `Media` asset also
calls the `FSII_CustomFlexFilter` filter, which takes the value of the input
attribute (`MediaCustomInput`) and inserts a derived string value into the
output attribute (`MediaCustomOutput`).

Figure 8-4   Inspect Dialog

8.3 Document Transformation Flex Filter

For your document transformation flex filter, Oracle WebCenter Sites provides a
default solution that converts documents into raw text files. If you need a flex filter

that converts documents into different formats, you can create a custom solution, register its components, and use them with your flex filter.

Implementing a Document Transformation flex filter requires the following components:

- A transformation engine that is registered in the `SystemTransforms` table and named to indicate the target file type; for example, `CS:Convert to Raw Text`.

- A document transformer, which is a custom class that performs document conversion (for example, converting binary files to raw text files). The document transformer class implements the following interface:

  `com.fatwire.transformer.common.DocumentTransformer`

- The `transformer-formats.xml` file, which is used to associate the transformation engine with the document transformer. The file, located in the WebCenter Sites `WEB-INF/classes` folder, specifies the target file type and the document transformer.

When the Document Transformation flex filter is invoked, the transformation engine functions as a wrapper. The engine forwards calls (using the `transformer-formats.xml` file) to the document transformer, which then performs file conversion.

See these topics:

- Default Solution

- About Custom Solutions

- Using a Default Transformation Engine

- Customizing Document Transformation Flex Filter

8.3.1 Default Solution

WebCenter Sites provides a default solution that can be used to convert documents to raw text files. The default components follow:

- The `CS:Convert to Raw Text` transformation engine, registered in the `SystemTransforms` table.

- A document transformer named `com.fatwire.transformer.tika.DocumentTransformerImpl`, which is coded to output raw text files once it is invoked by the `CS:Convert to Raw Text` engine.

- The `transformer-formats.xml` file, which is configured to associate the `CS:Convert to Raw Text` engine with the document transformer class named above.

Using the default solution requires you to implement a corresponding Document Transformation flex filter, which makes the transformation engine accessible from the WebCenter Sites interface as a document transformation option.

8.3.2 About Custom Solutions

To design a document transformation solution other than the default solution described above, create and customize the following components:

1. Write and deploy a document transformer for the target file type.

2. Register the transformation engine for the target file type.

3. Configure the `transformer-formats.xml` file to specify the document transformer and the target file type. (The xml file supports multiple document transformers.)

4. Implement the Document Transformation flex filter as described in this chapter.

See Customizing Document Transformation Flex Filter.

8.3.3 Using a Default Transformation Engine

WebCenter Sites provides the `CS:Convert` to Raw Text transformation engine. All of the engines are registered by default in the `SystemTransforms` table. If your document transformer is written to output files of type HTML, or HTML fragment, or XML, use the corresponding engine.

To use a default transformation engine:

1. Open Oracle WebCenter Sites Explorer.

2. Select the `SystemTransforms` table.

3. Locate the engine you have created. Note the value of the engine's **target** field. You enter this value for `<mime-type>` in the `transformer-formats.xml` file, discussed in Registering the Document Transformer.

4. In the **args** field, set the arguments that are appropriate for this transformation engine. For example: `exporttype=HTML`.

5. Continue to Registering the Document Transformer.

8.3.4 Customizing Document Transformation Flex Filter

WebCenter Sites provides a default document transformer, `com.fatwire.transformer.tika.DocumentTransformerImpl`, which is coded to output raw text files once it is invoked by the `CS:Convert to Raw Text` engine. For the target file other than raw text, create and register the flex filter's supporting components as follows.

Before implementing a Document Transformer flex filter:

- Writing and Deploying a Document Transformer Flex Filter

- Registering the Transformation Engine

- Registering the Document Transformer

8.3.4.1 Writing and Deploying a Document Transformer Flex Filter

You can write a deploy a document transformer flex filter

1. Write an implementation of the `com.fatwire.transformer.common.DocumentTransformer` interface. You will implement the following methods:

```
public String getOutputDocument(String filename,
   TransformerFormat outputformat);
```

and

```
public String getOutputDocument(String filename,String inputFileExt,
   TransformerFormat outputformat);
```

You can return `null` in the following method, as this method has been deprecated:

```
public InputStream getBytesAsStream(String filename,
   TransformerFormat outputformat);
```

2. Copy the document transformer's `jar` or `class` file to the WebCenter Sites web application `lib` folder or `classes` folder.

- For WebLogic Server:

   ```
   app-server-install-dir/bea/path-to-domain/domain-name/applications/WEB-INF/
   lib
   ```

- For WebSphere Application Server:

   ```
   WebSphere-Installation-Directory/InstalledApps/WEB-INF/lib
   ```

8.3.4.2 Registering the Transformation Engine

If you have written a document transformer to output files other than raw text, do the following procedure. Otherwise, see .

To register a transformation engine:

1. Open Oracle WebCenter Sites Explorer to add a row to the `SystemTransforms` table for the new transformation engine.

2. Select the `SystemTransforms` table and click in the table's workspace.

3. Select **File**, then **New**, and then **Record** and fill in the new row as follows:

   - In the **name** column, enter the name of the transformation engine.

   - In the **description** column, enter a description of the engine.

   - In the **target** column, enter `text/<filetype>`. You will use the target value for the `<mime-type>` at the time of registering the document transformer.

   - In the **classname** column, enter the engine's default class name: `com.fatwire.transformer.common.FWTransformer`.

   - In the **args** column, set any arguments that are appropriate for this transformation engine.

4. Save your changes.

8.3.4.3 Registering the Document Transformer

You can register your document transformer in the `transformer-formats.xml` file, located in the WebCenter Sites `WEB-INF/classes` folder. The default file looks like this:

```
<transformer-format>
<name>Text Format</name>
<!-- name of the output format supported by this repository -->
<mime-type>text/plain</mime-type>
```

```
<file-extension>txt</file-extension>
<transformer-options>
<!-- number of possible transformers available for this transformation -->
  <transformer>
      <name>TIKA</name>
        <properties>
          <property>
            <name>ClassName</name>
<!-- name of the transformer class that gets loaded by transformer factory -->
            <value>com.fatwire.transformer.tika.DocumentTransformerImpl</value>
          </property>
          <property>
            <name>InputFile_Exts</name>
<!-- allowed input file extensions..* means all file types supported by tika -->
            <value>*</value>
          </property>
        </properties>
  </transformer>
</transformer-options>
</transformer-format>
```

1. Set the `<mime-type>` to the value of the **target** field for the transformation engine.

   The value for `target` is set when registering the transformation engine.

2. Specify the class name of your document transformer.

   The class name is created when writing and deploying a document transformer flex filter.

3. To specify multiple document transformers, repeat the entries for each transformer.

   The value for `<mime-type>` determines which document transformer will be invoked by the transformation engine.

8.4 SampleFlexFilter.java

A working example of a custom flex filter class that extends the `AbstractFlexFilter` class, and whose purpose is to access a flex attribute and set a derived attribute value.

```
/**
Copyright (c) 2010 Oracle Corporation. All Rights Reserved. Title,
ownership
rights, and intellectual property rights in and to this software remain
with
FatWire Corporation. This software is protected by international copyright
laws and treaties, and may be protected by other law. Violation of
copyright
laws may result in civil liability and criminal penalties.
*/
package com.fatwire.firstsite.filter;
.
import java.util.Enumeration;
import java.util.Set;
.
.
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
.
import COM.FutureTense.Interfaces.FTValList;
.
```

```
import com.openmarket.basic.interfaces.AssetException;
import com.openmarket.basic.interfaces.IListBasic;
import com.openmarket.gator.flexfilters.AbstractFlexFilter;
import com.openmarket.gator.interfaces.IFilterDependencies;
import com.openmarket.gator.interfaces.IFilterDescription;
import com.openmarket.gator.interfaces.IFilterEnvironment;
import com.openmarket.gator.interfaces.IFilterableAssetInstance;
.
/**
Sample Flex Filter class.
This filter class demonstrates a very simple filter application. It takes
an incoming
Flex asset attribute and creates a derived attribute from it.
The derived attribute that is generated by this filter contains the filter
identifier,
assetId/assetType of the asset being changed, and the value of the asset
itself.
It also writes an informational log entry with various characteristics of
the asset
being saved.
The intent is not to provide a useful filter but simply to demonstrate the
mechanics of building a custom filter.
*
*/
public class SampleFlexFilter extends AbstractFlexFilter {
// This defines the input and output attribute names. These values are
// overridden by the sample constructor when there is an initialization
string
// provided by the database entry for the filter. The database
initialization string
// is optional.
private static String ARG_CUSTOM[] =
{ "Input custom string" , "Output custom string" }
;
private static final Log log= LogFactory.getLog(LoggerPropDesc.LOG_NAME);
.
/**
A filter implementation is required to have a single-argument public
constructor that expects a COM.FutureTense.Interfaces.FTValList
object.
The FTValList provided to the constructor is the set of configuration
arguments that were registered in the database entry for the filter.
*
@param configurationParameters
*
This constructor will override statically defined arguments within
this
filter with values taken from the database. If nothing is defined in
the
database filter definitions then the static values will prevail.
*
*/
@SuppressWarnings("unchecked")
public SampleFlexFilter(FTValList configurationParameters) {
super(configurationParameters);
if ( configurationParameters == null || configurationParameters.size() <
1 )
{ log.info("SampleFlexFilter constructor: there are no arguments in Filters table
entry"); }
else {
// Bingo! Something is coming from the database so break it down
```

```
// and replace the static variables.
Set<String> entries = configurationParameters.keySet();
ARG_CUSTOM = new String[entries.size()];
int i = 0;
for ( String key : entries )
{ String value = configurationParameters.getValString(key);
log.info("SampleFlexFilter constructor : "+key+"="+value); ARG_CUSTOM[i++] = new
String(key); }
}
}
.
/**
Perform the filter operation. This method is entered after an asset is
created or
modified. There is no way to reject the operation. All changes occur to
the
IFilterableAssetInstance.
*/
@Override
public void filterAsset(IFilterEnvironment env, String filterIdentifier,
FTValListfilterArguments, IFilterableAssetInstance instance)
throws AssetException {
// Get the identifier and name of the asset being saved.
String assetid = instance.getAssetID();
String assetname = instance.getName();
// Put some interesting information in the log.
log.info(filterIdentifier+" SampleFlexFilter filterAsset :"
+" FilterType="+env.getFilterType()
+", AttributeType="+env.getAttributeType()
+", AssetTypeName="+instance.getAssetTypeName()
+", GroupTypeName="+instance.getGroupTypeName()
);
// Setup the variables to generate the derived attribute.
String assetValue = "FilterId="filterIdentifier",
assetId="assetid"("assetname")";
String value = " <noInput>";
.
String inputattr = getAttrID(env, filterArguments, ARG_CUSTOM[0]);
IListBasic ilistbasic = instance.getAttribute(inputattr);
if ( inputattr != null && ilistbasic != null && ilistbasic.hasData() ) {
try {
value = " ["+ilistbasic.getValue("value")+"]";
}
catch ( NoSuchFieldException e) {
log.info("SampleFlexFilter : NoSuchFieldException");
}
}
.
// Create the derived attribute using the attribute name and the string
values
// that were initialized above.
instance.addDerivedDataValue(filterIdentifier
,
env.getAttributeIdentifier(filterArguments.getValString(ARG_CUSTOM[1]))
, assetValue+value);
}
.
/**
Describe all the potential derived attributes, parent affinities, and
recommendations the filter might set.
*
```

```
This method exists as part of the AbstractFlexFilter class and
does not need to be implemented in a custom filter unless the default
behavior
needs to be changed. The default is to describe all attributes arguments
as
eligible for being set by this filter.
*/
@Override
public void describeDerivedAttributes(IFilterEnvironment env,
String filterIdentifier, FTValList filterArguments,
String defTypeName, String parentDefTypeName,
IFilterDescription descriptionObject) throws AssetException {
String attrname =
env.getAttributeIdentifier(filterArguments.getValString(ARG_CUSTOM[1]));
if (attrname != null)
descriptionObject.addAttribute(filterIdentifier, attrname, false,
true);
}
.
/**
Describes the filter's asset dependencies. This sample implementation
uses 'exist' dependency.
*
This method exists as part of the AbstractFlexFilter class and
does not need to be implemented in a custom filter unless the default
behavior
needs to be changed.
*/
@Override
public void getDependencies(IFilterEnvironment env,
String filterIdentifier, FTValList filterArguments,
String assetTypeName, String parentTypeName,
IFilterDependencies filterdeps) throws AssetException {
.
Enumeration<?> args = getLegalArguments(env, filterIdentifier).keys();
while(args.hasMoreElements())
Unknown macro: {String currentAttrId = getAttrID(env, filterArguments,
(String)args.nextElement()); if (currentAttrId != null && currentAttrId.length() >
0) { // Exists or exact filterdeps.addExistsToDeps(env.getAttributeType(),
currentAttrId); } }
}
.
/**
Return a list of legal filter arguments. This method is called by the CS
Advanced UI
to display the list of valid arguments accepted by the filter.
*/
@Override
public FTValList getLegalArguments(IFilterEnvironment env,
String filterIdentifier) throws AssetException {
.
FTValList ftvallist = new FTValList();
ftvallist.setValString(ARG_CUSTOM[0], ARG_CUSTOM[0]);
ftvallist.setValString(ARG_CUSTOM[1], ARG_CUSTOM[1]);
return ftvallist;
}
.
/**
Obtain the legal values for a single filter argument. By
returning null indicates that any value is legal. Implementing this
method is optional. The default is to allow any value.
```

```
*/
@Override
public String[] getArgumentLegalValues(IFilterEnvironment env,
String filterIdentifier, String argumentName) throws AssetException {
.
String[] legalValues = null;
if ( argumentName.equalsIgnoreCase(ARG_CUSTOM[0]) ) {
legalValues = new String[3];
legalValues[0]= "FSII_CustomInput";
legalValues[1] = "CustomInput";
legalValues[2] = "InputAttribute";
}
else if ( argumentName.equalsIgnoreCase(ARG_CUSTOM[1]) ) {
legalValues = new String[1];
legalValues[0] = "FSII_CustomOutput";
}
return legalValues;
}
}
```

9

Designing Attribute Editors

Attribute editors are displayed in flex and flex parent assets' New and Edit forms. Through attribute editor you define how users will enter attribute data. To help you create attribute editors, WebCenter Sites provides the presentationobject.dtd file that defines the input types, the attribute editor asset whose XML code provides the values of the input type's options such as check boxes, radio button, and attribute editor elements that receive input values and supply the logic behind the format and behavior of the attribute.

Topics:

- About Attribute Editors

- Creating Attribute Editors

- Customizing Attribute Editors

- Considerations About Editing Attribute Editors

9.1 About Attribute Editors

Attribute editors are like any other assets, so you edit, approve, and manage them the way you would any other asset. Through attribute editors you can define how users enter data for attributes on New or Edit forms for flex or flex parent assets.

When you assign an attribute editor to an attribute, it replaces the default input mechanism (style) for that attribute. The default input style is based on the data type of the attribute. Attribute editors format the input mechanism for attributes, so you design your attribute editors as you design your flex attributes. You can use the workflow and revision tracking features to manage attribute editors.

There are three parts to an attribute editor, with an optional fourth and fifth:

- The `presentationobject.dtd` file, located in the WebCenter Sites installation directory (Required). This is the DTD file that defines all the possible input styles (presentation objects) for flex attributes and their style tags.

- The attribute editor asset (Required). It holds or points to XML code that provides input options for the attribute it is associated with. You use the style tags defined in the DTD to create this XML code.

- An element that formats the attribute, or, displays an edit mechanism, when that attribute displays in a New or Edit form (Required). This element must be located in the `OpenMarket/Gator/AttributeTypes` directory in the `ElementCatalog` table to be able to find it. Its name must exactly match the name of the style tag that invokes it from the attribute editor.

- An element that formats the attribute value when it displays in an Inspect form (Optional). This element must also be located in the `OpenMarket/Gator/AttributeTypes` directory in the `ElementCatalog` table.

  The name of the element must use the convention `DisplayStyleTag`, where `StyleTag` represents and must exactly match the name of the style tag that invokes it from the attribute editor.

- An element that formats the attribute data before it is saved in the database (Optional). This element must also be located in the `OpenMarket/Gator/AttributeTypes` directory in the `ElementCatalog` table.

  The name of the element must use the convention `StyleTagFlexAssetGather`, where `StyleTag` represents and must exactly match the name of the style tag that invokes it from the attribute editor.

WebCenter Sites provides the following items, by default, to support the development of your attribute editors:

- The `presentationobject.dtd` file. It defines several input styles (presentation objects) that you can use in your attribute editors. This means you do not have to define your own unless the nine that are included do not cover your needs.

- Nine text files with sample XML that you can use to create attribute editor assets. You can cut and paste the sample XML into your attribute editor assets. These files are located in the installation directory under `Samples/Attribute_Editors`.

- Ten display elements that work with the sample XML code for attribute editor assets. They are located in the `OpenMarket/Gator/AttributeTypes` directory in the `ElementCatalog` table.

When you do not use attribute editors, WebCenter Sites uses default input styles for the attributes, based on their data types. See Default Input Styles for Attributes. Create attribute editors only when the default input styles are not sufficient for your attributes.

Topics:

- The presentationobject.dtd File

- The Attribute Editor Asset

- The Attribute Editor Elements

- Conventions for the Attribute Editor Elements

9.1.1 The presentationobject.dtd File

The `presentationobject.dtd` file defines all of the input types (presentation objects) that you can implement through attribute editors. The default `presentationobject.dtd` file defines nine input style tags and the arguments that they can pass from the attribute editor to the display elements (described in The Attribute Editor Elements).

Following is the entire `presentationobject.dtd` file. It is located in the WebCenter Sites installation directory:

```
<!-- PRESENTATIONOBJECT: An editor
-- PRESENTATIONOBJECT defines the presentation object for
-- instances of Gator attribute types. A presentation object
```

```
-- defines the properties of an editor for one of the following
-- controls: Text field, Text area, Pulldown menu
-- For additional information, refer to
-- com.openmarket.gator.interfaces.IPresentationObject.
-- You must specify one of TEXTFIELD, TEXTAREA, or PULLDOWN
-- elements.
-->
<!ELEMENT PRESENTATIONOBJECT (TEXTFIELD | TEXTAREA | PULLDOWN |
RADIOBUTTONS | CHECKBOXES | PICKASSET | FIELDCOPIER | DATEPICKER |
IMAGEPICKER | CKEDITOR | FCKEDITOR | IMAGEEDITOR | TYPEAHEAD |
UPLOADER)>
<!ATTLIST PRESENTATIONOBJECT NAME CDATA #REQUIRED>
<!-- TEXTFIELD: A text field of a specific width
-- You must specify the x dimension; the maximum number of
-- allowable characters defaults to 255.
-->
<!ELEMENT TEXTFIELD ANY>
<!ATTLIST TEXTFIELD XSIZE CDATA #IMPLIED>
<!ATTLIST TEXTFIELD WIDTH CDATA #IMPLIED>
<!ATTLIST TEXTFIELD MAXCHARS CDATA "255">
<!ATTLIST TEXTFIELD BLANKED (YES | NO) "NO">
<!ATTLIST TEXTFIELD MAXVALUES CDATA #IMPLIED>
<!-- TEXTAREA: A text area of a specific size
-- The wrap style defaults to soft.
-->
<!ELEMENT TEXTAREA ANY>
<!ATTLIST TEXTAREA XSIZE CDATA #IMPLIED>
<!ATTLIST TEXTAREA YSIZE CDATA #IMPLIED>
<!ATTLIST TEXTAREA WIDTH CDATA #IMPLIED>
<!ATTLIST TEXTAREA HEIGHT CDATA #IMPLIED><!ATTLIST TEXTAREA WRAPSTYLE (OFF | SOFT |
HARD) "SOFT">
<!ATTLIST TEXTAREA DEPTYPE CDATA #IMPLIED>
<!ATTLIST TEXTAREA MAXVALUES CDATA #IMPLIED>
<!ATTLIST TEXTAREA RESIZE CDATA #IMPLIED>
<!-- PULLDOWN: A pulldown menu with an enumeration of items
-- You can specify zero or more list items; the fontsize defaults
-- to relative fontsize 3.
-->
<!ELEMENT PULLDOWN ((ITEM)* | QUERYASSETNAME)>
<!ATTLIST PULLDOWN FONTSIZE CDATA #IMPLIED>
<!-- RADIOBUTTONS: Radio buttons with an enumeration of items
-- You can specify zero or more list items; the fontsize defaults
to relative fontsize 3.
-->
<!ELEMENT RADIOBUTTONS ((ITEM)* | QUERYASSETNAME)>
<!ATTLIST RADIOBUTTONS FONTSIZE CDATA #IMPLIED>
<!ATTLIST RADIOBUTTONS LAYOUT (HORIZONTAL | VERTICAL) "VERTICAL">
<!-- CHECKBOXES: Check boxes with an enumeration of items
-- You can specify zero or more list items; the fontsize defaults
-- to relative fontsize 3.
-->
<!ELEMENT CHECKBOXES ((ITEM)* | QUERYASSETNAME)>
<!ATTLIST CHECKBOXES FONTSIZE CDATA #IMPLIED>
<!ATTLIST CHECKBOXES LAYOUT (HORIZONTAL | VERTICAL) "HORIZONTAL">
<!-- ITEM: A list item
-- You can specify zero or more characters of text.
-->
<!ELEMENT ITEM (#PCDATA)*>
<!-- SQL: Query to populate list of items
-- You can specify zero or more characters of text. Query must
-- return a 'value' column.
```

```
                        -->
                        <!ELEMENT QUERYASSETNAME (#PCDATA)*>
                        <!ELEMENT CKEDITOR ANY>
                        <!ATTLIST CKEDITOR ALLOWEDASSETTYPES CDATA #IMPLIED>
                        <!ATTLIST CKEDITOR SCRIPT CDATA #IMPLIED>
                        <!ATTLIST CKEDITOR IMAGEPICKERID CDATA #IMPLIED>
                        <!ATTLIST CKEDITOR IMAGEASSETTYPE CDATA #IMPLIED>
                        <!ATTLIST CKEDITOR TOOLBAR CDATA #IMPLIED>
                        <!ATTLIST CKEDITOR DEPTYPE CDATA #IMPLIED>
                        <!ATTLIST CKEDITOR WIDTH CDATA #IMPLIED>
                        <!ATTLIST CKEDITOR HEIGHT CDATA #IMPLIED>
                        <!ATTLIST CKEDITOR MAXVALUES CDATA #IMPLIED>
                        <!ATTLIST CKEDITOR RESIZE CDATA #IMPLIED>
                        <!ATTLIST CKEDITOR CONFIG CDATA #IMPLIED>
                        <!ATTLIST CKEDITOR CONFIGOBJ CDATA #IMPLIED>
                        <!-- Deprecated in Oracle WebCenter
                        Sites 11gR1 release. -->
                        <!ELEMENT FCKEDITOR ANY>
                        <!ATTLIST FCKEDITOR XSIZE CDATA #IMPLIED>
                        <!ATTLIST FCKEDITOR YSIZE CDATA #IMPLIED>
                        <!ATTLIST FCKEDITOR LAZYLOAD CDATA #IMPLIED>
                        <!ATTLIST FCKEDITOR ALLOWEDASSETTYPES CDATA #IMPLIED>
                        <!ATTLIST FCKEDITOR SCRIPT CDATA #IMPLIED>
                        <!ATTLIST FCKEDITOR IMAGEPICKERID CDATA #IMPLIED>
                        <!ATTLIST FCKEDITOR TOOLBAR CDATA #IMPLIED>
                        <!ATTLIST FCKEDITOR DEPTYPE CDATA #IMPLIED>
                        <!ATTLIST FCKEDITOR WIDTH CDATA #IMPLIED>
                        <!ATTLIST FCKEDITOR HEIGHT CDATA #IMPLIED>
                        <!ATTLIST FCKEDITOR MAXVALUES CDATA #IMPLIED>
                        <!-- PICKASSET: When the tree is active, it's the "add from tree"
                        -- button.
                        -- When the tree is disabled, it's The Content Centre remember
                        -- widget. -->
                        <!ELEMENT PICKASSET ANY>
                        <!ATTLIST PICKASSET MAXVALUES CDATA #IMPLIED>
                        <!ATTLIST PICKASSET DISPLAYELEMENT CDATA #IMPLIED>
                        <!-- TYPEAHEAD: Type And Select the asset. -->
                        <!ELEMENT TYPEAHEAD ANY>
                        <!ATTLIST TYPEAHEAD MAXVALUES CDATA #IMPLIED>
                        <!ATTLIST TYPEAHEAD DISPLAYELEMENT CDATA #IMPLIED>
                        <!ATTLIST TYPEAHEAD PAGESIZE CDATA #IMPLIED>
                        <!-- UPLOADER: Upload a file from local disc. -->
                        <!ELEMENT UPLOADER ANY>
                        <!ATTLIST UPLOADER MAXVALUES CDATA #IMPLIED>
                        <!ATTLIST UPLOADER MAXFILESIZE CDATA #IMPLIED>
                        <!ATTLIST UPLOADER FILETYPES CDATA #IMPLIED>
                        <!ATTLIST UPLOADER MINWIDTH CDATA #IMPLIED>
                        <!ATTLIST UPLOADER MAXWIDTH CDATA #IMPLIED>
                        <!ATTLIST UPLOADER MINHEIGHT CDATA #IMPLIED>
                        <!ATTLIST UPLOADER MAXHEIGHT CDATA #IMPLIED>
                        <!-- FIELDCOPIER: A hidden field whose value is set from another
                        -- field.
                        -- ex. If you want an attribute whose value is always the name of
                        -- the asset:
                        -- <FIELDCOPIER SOURCEFIELD="name"/>
                        -->
                        <!ELEMENT FIELDCOPIER ANY>
                        <!ATTLIST FIELDCOPIER SOURCEFIELD CDATA #REQUIRED>
                        <!-- This describe the Date Picker -->
                        <!ELEMENT DATEPICKER ANY>
                        <!ATTLIST DATEPICKER COMPARETOFIELD CDATA #REQUIRED>
```

```
<!ATTLIST DATEPICKER MAXVALUES CDATA #IMPLIED>
<!-- This describe the ImagePicker -->
<!ELEMENT IMAGEPICKER ANY>
<!ATTLIST IMAGEPICKER ASSETTYPENAME CDATA #REQUIRED>
<!ATTLIST IMAGEPICKER ATTRIBUTETYPENAME CDATA #REQUIRED>
<!ATTLIST IMAGEPICKER ATTRIBUTENAME CDATA #REQUIRED>
<!ATTLIST IMAGEPICKER CATEGORYATTRIBUTENAME CDATA #IMPLIED>
<!ATTLIST IMAGEPICKER RESTRICTEDCATEGORYLIST CDATA #IMPLIED>
<!ATTLIST IMAGEPICKER MAXVALUES CDATA #IMPLIED>
<!-- Image Editor -->
<!ELEMENT IMAGEEDITOR ANY>
<!ATTLIST IMAGEEDITOR EDITORTYPE (oie | clarkii) "oie">
<!ATTLIST IMAGEEDITOR HEIGHT CDATA #REQUIRED>
<!ATTLIST IMAGEEDITOR WIDTH CDATA #REQUIRED>
<!ATTLIST IMAGEEDITOR FITIMAGE (true | false) "true">
<!ATTLIST IMAGEEDITOR SNAPSHOTPANEL (true | false) "false">
<!ATTLIST IMAGEEDITOR LIMITCROPPING (true | false) "false">
<!ATTLIST IMAGEEDITOR CROPHEIGHT CDATA #IMPLIED>
<!ATTLIST IMAGEEDITOR CROPWIDTH CDATA #IMPLIED>
<!ATTLIST IMAGEEDITOR ENABLEOIEFORMAT (true | false) "false">
<!ATTLIST IMAGEEDITOR LIMITSIZE (true | false) "false">
<!ATTLIST IMAGEEDITOR MAXHEIGHT CDATA #IMPLIED>
<!ATTLIST IMAGEEDITOR MAXWIDTH CDATA #IMPLIED>
<!ATTLIST IMAGEEDITOR MINHEIGHT CDATA #IMPLIED>
<!ATTLIST IMAGEEDITOR MINWIDTH CDATA #IMPLIED>
<!ATTLIST IMAGEEDITOR AUTORESAMPLE (true | false) "false">
<!ATTLIST IMAGEEDITOR AUTORESAMPLEPROPORTIONAL (true | false)
"false">
<!ATTLIST IMAGEEDITOR DEFAULTTEXTFONT CDATA #IMPLIED>
<!ATTLIST IMAGEEDITOR DEFAULTTEXTSIZE CDATA #IMPLIED>
<!ATTLIST IMAGEEDITOR DEFAULTTEXTCOLOR CDATA #IMPLIED>
<!ATTLIST IMAGEEDITOR ASSETTYPE CDATA #IMPLIED>
<!ATTLIST IMAGEEDITOR ATTRIBUTE CDATA #IMPLIED>
<!ATTLIST IMAGEEDITOR ATTRIBUTETYPE CDATA #IMPLIED>
<!ATTLIST IMAGEEDITOR CATEGORYATTRIBUTE CDATA #IMPLIED>
<!ATTLIST IMAGEEDITOR RESTRICTEDCATEGORYLIST CDATA #IMPLIED>
<!ATTLIST IMAGEEDITOR ENABLEIMAGEPICKER (true | false) "false">
<!ATTLIST IMAGEEDITOR OIEASSETTYPE CDATA #IMPLIED>
<!ATTLIST IMAGEEDITOR OIEATTRIBUTE CDATA #IMPLIED>
<!ATTLIST IMAGEEDITOR OIEATTRIBUTETYPE CDATA #IMPLIED>
<!ATTLIST IMAGEEDITOR OIECATEGORYATTRIBUTE CDATA #IMPLIED>
<!ATTLIST IMAGEEDITOR OIERESTRICTEDCATEGORYLIST CDATA #IMPLIED>
<!ATTLIST IMAGEEDITOR OIEENABLEIMAGEPICKER (true | false) "false">
<!ATTLIST IMAGEEDITOR TAGEDIT (true | false) "false">
<!ATTLIST IMAGEEDITOR BASE64JPEGQUALITY CDATA "95">
<!ATTLIST IMAGEEDITOR ASKTOSAVELOCALLY (true | false) "false">
<!ATTLIST IMAGEEDITOR DEFAULTSAVINGTYPE (gif | jpg | jpe | png |
tif | bmp | oie) "gif">
<!ATTLIST IMAGEEDITOR ENABLEGIFSAVING (true | false) "true">
<!ATTLIST IMAGEEDITOR ENABLEJPEGSAVING (true | false) "true">
<!ATTLIST IMAGEEDITOR ENABLEPNGSAVING (true | false) "true">
<!ATTLIST IMAGEEDITOR ENABLETIFFSAVING (true | false) "true">
<!ATTLIST IMAGEEDITOR ENABLEBMPSAVING (true | false) "true">
<!ATTLIST IMAGEEDITOR GRIDVISIBLE (true | false) "false">
<!ATTLIST IMAGEEDITOR GRIDSNAP (true | false) "true">
<!ATTLIST IMAGEEDITOR GRIDSPACINGX CDATA #IMPLIED>
<!ATTLIST IMAGEEDITOR GRIDSPACINGY CDATA #IMPLIED>
<!ATTLIST IMAGEEDITOR MAXTHUMBNAILHEIGHT CDATA #IMPLIED>
<!ATTLIST IMAGEEDITOR MAXTHUMBNAILWIDTH CDATA #IMPLIED>
<!ATTLIST IMAGEEDITOR THUMBNAILFORMAT (gif | jpg | jpe | png | tif
```

```
| bmp | oie) "gif">
<!ATTLIST IMAGEEDITOR MAXVALUES CDATA #IMPLIED>
```

To create custom attribute editors other than the ones made possible by default, you must first define an XML input style tag, a PRESENTATIONOBJECT tag, in the presentationobject.dtd file. To define a new PRESENTATIONOBJECT tag, you must do the following:

- Add the new tag (presentation object) to the list in the <!ELEMENT PRESENTATIONOBJECT...> statement.

- Add a <!ELEMENT...> section that defines the new tag (presentation object) and the arguments that it takes. Follow the normal syntax rules for a .dtd file and follow the conventions used in the presentationobject.dtd file.

9.1.2 The Attribute Editor Asset

The attribute editor asset either holds XML code or points to an .xml file. That XML code provides the values of the input type's options such as check boxes, radio options, drop-down lists, and so on. Although WebCenter Sites provides nine text files with sample code that you can use to create attribute editor assets, it does not provide any attribute editor assets because you have to customize the sample code so that any options are appropriate for your data.

When you create your attribute editors, either cut and paste the code from HTML version of this book (samples follow this section) or use the text files located in the Samples subdirectory of the installation directory on your system.

This section includes the following topics:

- The Syntax and the Default Tags

- CHECKBOXES

- CKEditor

- PICKASSET

- PULLDOWN Example

- RADIOBUTTONS

- TEXTAREA

- TEXTFIELD

- TYPEAHEAD

- UPLOADER

9.1.2.1 The Syntax and the Default Tags

The code in an attribute editor asset has the following basic format:

```
<?XML VERSION="1.0"?>
<!DOCTYPE PRESENTATIONOBJECT SYSTEM "presentationobject.dtd">

<PRESENTATIONOBJECT NAME="SomeName">
...
...
...
```

```
</PRESENTATIONOBJECT>
```

The tag that describes the format of the input style (presentation object) is embedded between the pair of PRESENTATIONOBJECT tags. This tag can have additional nested tags in it. Although the NAME attribute is required for the PRESENTATIONOBJECT tag, it is reserved for future use.

The name of any PRESENTATIONOBJECT tag that you include in the code for an attribute editor asset must be defined in the presentationobject.dtd file. This .dtd file has the following PRESENTATIONOBJECT tags defined by default (listed in alphabetic order):

- CHECKBOXES

- CKEDITOR

- DATEPICKER

- FIELDCOPIER

- IMAGEEDITOR

- IMAGEPICKER

- PICKASSET

- PICKFROMTREE (Deprecated; use PICKASSET instead.)

- PULLDOWN

- RADIOBUTTONS

- TEXTAREA

- TEXTFIELD

- TYPEAHEAD

- UPLOADER

Note that the PRESENTATIONOBJECT tag that you use in the attribute editor code must exactly match the name of the display element that you want to use for the attribute editor. Therefore, to define a new tag for a custom attribute editor, the element that you create must use the same name as the tag.

For a description of the elements, see The Attribute Editor Elements. For code samples for attribute editors, see the following sections:

- CHECKBOXES

- CKEditor

- PICKASSET

- PULLDOWN Example

- RADIOBUTTONS

- TEXTAREA

- TEXTFIELD

- [TYPEAHEAD](#)

- [UPLOADER](#)

9.1.2.2 CHECKBOXES

The `presentationobject.dtd` defines a `CHECKBOXES` tag, an attribute editor that uses the tag to invoke the `CHECKBOXES` element, which creates a set of check boxes for the attribute.

The `CHECKBOXES` tag takes the following parameters:

- `ITEM` or `QUERYASSETNAME`: The source of the names listed next to the check boxes. To specify the names, use the `ITEM` parameter. To specify a query asset that obtains the names dynamically from a database table, use the `QUERYASSETNAME` parameter.

  Note the following:

  - Whether you use the `ITEM` parameter or the `QUERYASSETNAME` parameter as the source of the names listed next to the check boxes, ensure the parameter specifies multiple values. A single value creates radio buttons instead of check boxes.

  - You must use a query asset to use a query. You cannot use a SQL statement.

  - The SQL in the query asset must return a value column. For example: `select name as value from shippingtype`.

  - If the data type of the attribute using the attribute editor is asset, the query must also return the assets IDs. For example: `select name as value, id as assetid from Product where...`

- `LAYOUT`: Whether the check boxes should be positioned in a vertical list or spread out in a horizontal row. Valid options are `HORIZONTAL` or `VERTICAL`. The default is `HORIZONTAL`.

- `MAXVALUES`: Limits the number of values that can be added to a multi-valued attribute. For example, if `MAXVALUES` is set to `10`, then only ten values can be added. No default value.

The following attribute editor code specifies that the `CHECKBOXES` element should use the results of a query asset named `A Prods` for the names of a vertical list of check boxes:

```
<?XML VERSION="1.0"?>
<!DOCTYPE PRESENTATIONOBJECT SYSTEM "presentationobject.dtd">
<PRESENTATIONOBJECT NAME="CheckBox">
 <CHECKBOXES LAYOUT="VERTICAL">
  <QUERYASSETNAME>A Prods</QUERYASSETNAME>
 </CHECKBOXES>
</PRESENTATIONOBJECT>
```

For example code that shows the use of the `ITEM` parameter, see PULLDOWN Example.

Notes about data types:

A `CHECKBOXES` attribute editor is appropriate for attributes with the following data types:

- `date`

- `float`

- `integer`

- `money`

- `string`

- `asset` (If asset, you must supply the name of the query asset that returns the names of the assets.)

9.1.2.3 CKEditor

The `presentationobject.dtd` defines a `CKEDITOR` tag. An attribute editor that uses the tag invokes the `CKEDITOR` element, which launches the CKEditor. The person creating the flex asset enters the value for the attribute in that window.

- You must have the CKEditor application installed and configured correctly.

- It is highly recommended that you use CKEditor only when the data type of the attribute is set to `blob`. You don't have to worry about sizing the field for the `blob` data type.

The `CKEDITOR` tag takes the following parameters:

The following code includes a `CKEDITOR` tag that creates a text box that is 400 pixels wide by 200 pixels high:

- `EMBEDDEDLINKS`: The `YES` value lets users create links to other assets. Default value: `NO`.

- `ALLOWEDASSETTYPES`: Lets you define which asset types can be included or linked to by means of CKEditor. Default value: `ALL`.

- `DEPTYPE`: The approval dependency between the main asset and an embedded asset. Valid options are `EXISTS` and `EXACT`. Default value: `EXACT`.

- `MAXVALUES`: This parameter limits the number of values that can be added to a multi-valued attribute. For example, if `MAXVALUES` is set to `10`, then only ten values can be added. No default value.

- `MAXLENGTH`: The number of characters in the CKEditor's source view (displayed when you click the CKEditor **Source** button). Default value: `-1` (unlimited).

- `INSTRUCTION`: Used to provide help for the field.

The `CKEDITOR` tag also takes parameters for customizing the appearance of CKEditor instances. One set of parameters is defined in WebCenter Sites. The other set is available on the CKEditor website.

Parameters defined in WebCenter Sites must be added directly to the **XML** field of a CKEditor instance:

- `TOOLBAR`: The name of the toolbar, which is read from the configuration file specified in the `CONFIG` parameter. For example, `CONFIG="myconfig.js"`.

- `WIDTH`: The text box width in pixels. If you do not specify a value for `WIDTH`, WebCenter Sites sets a default width. Sample value: `700px`.

- HEIGHT: The text box height in pixels. If you do not specify a value for HEIGHT, WebCenter Sites sets a default height. Sample value: 300px.

  > **Note:**
  >
  > The WIDTH and HEIGHT parameters replace the deprecated XSIZE and YSIZE parameters. Setting XSIZE and YSIZE has no effect.

- RESIZE: The true value allows users to resize the text area. Default value: false.

Parameters listed on the CKEditor website must be specified either in a configuration file, named in the CONFIG parameter, or in the CONFIGOBJ parameter. Except for the TOOLBAR parameter, they are not recognized when they are added to the CKEditor's **XML** field. The following is a description of the CONFIG and CONFIGOBJ parameters:

- CONFIG: Specifies the relative path to the default or custom CKEditor configuration file. For example, setting CONFIG="myconfig.js" will load the custom myconfig.js file from the CKEditor base path (CKEditor source folder). For a custom configuration file, users can also specify a URL to the config.js file in the CONFIG attribute. The base path is defined in the xcelerate.ckeditor.basepath property, in the wcs_properties.json file.

  Within the configuration file is a CKEDITOR.editorConfig function, where you specify the CKEditor parameters listed at http://docs.cksource.com/ckeditor_api/symbols/CKEDITOR.html#.editorConfig. The list includes the TOOLBAR parameter.

- CONFIGOBJ: Takes a JSON string, which can contain any of the parameters listed at http://docs.cksource.com/ckeditor_api/symbols/CKEDITOR.config.html. (The list includes the TOOLBAR parameter.)

  The JSON string format uses double quotes:

  ```
  '{foo1:"value1",foo2:"value2"}' ...
  ```

  For example:

  ```
  CONFIGOBJ='{width:"300px",height:"300px",toolbar:"SITES",fullPage:true}'
  ```

> **Note:**
>
> CKEditor parameters are recognized and used in the following hierarchical order:
>
> Parameters in the CKEditor's **XML** field override parameters in CONFIGOBJ, which override parameters in the configuration file named in the CONFIG parameter. For example, the TOOLBAR parameter in a CKEditor's **XML** field overrides the toolbar parameter in CONFIGOBJ='{toolbar: "SITES",fullPage:true}', which overrides the toolbar parameter in the configuration file myconfig.js (specified as CONFIG='myconfig.js'). The fullPage parameter in CONFIGOBJ overrides the same parameter in myconfig.js.
>
> To avoid potential conflicts, specify each parameter only once in the place where it is recognized: either in the editor's **XML** field, in the CONFIGOBJ parameter, or in the configuration file that is specified in the CONFIG parameter.

The following code includes a CKEDITOR tag that creates a CKEditor box 400 pixels wide by 200 pixels high:

```
<?XML VERSION="1.0"?>
<!DOCTYPE PRESENTATIONOBJECT SYSTEM "presentationobject.dtd">

<PRESENTATIONOBJECT NAME="CKEditorTest">
  <CKEDITOR WIDTH="400px" HEIGHT="200px">
  </CKEDITOR>
</PRESENTATIONOBJECT>
```

Notes about data types:

The best choice for the data type of an attribute that uses a CKEDITOR attribute editor is blob. You can use string or text but it is problematic because it is hard to predict how large the data entered into the attribute's field will be because each HTML marker counts toward the limit. The string data type is limited to 256 characters and text is limited to 2000. It is recommended that you use blob as the data type for attributes that use CKEDITOR as their input mechanism.

9.1.2.4 PICKASSET

The presentationobject.dtd defines a PICKASSET tag. An attribute editor that uses the tag invokes the PICKASSET element, which creates a **Drop Zone** field into which a user can drag and drop an asset from a tree or search results list. When an asset is dropped into the **Drop Zone** field, the name of the asset is displayed in a box outlined with a dashed border. When you point to the name of the asset, a tooltip opens displaying information about the asset. To change the behavior of the PICKASSET attribute editor, for example, to customize the look and feel of the tooltip, see Building an Attribute Editor.

This tag can take the MAXVALUES parameter. The MAXVALUES parameter limits the number of values that can be added to a multi-valued attribute. For example, if MAXVALUES is set to 10, then only ten values can be added. No default value.

For information about customizing the PICKASSET attribute editor, see Building an Attribute Editor.

Below is the code to create a PICKASSET attribute editor:

```
<?XML VERSION="1.0"?>
<!DOCTYPE PRESENTATIONOBJECT SYSTEM "presentationobject.dtd">
<PRESENTATIONOBJECT NAME="PickAsset">

  <PICKASSET>
  </PICKASSET>
</PRESENTATIONOBJECT>
```

Notes about data types:

A PICKASSET attribute editor is only appropriate for attributes with a data type of asset.

9.1.2.5 PULLDOWN Example

The presentationobject.dtd defines a PULLDOWN tag. An attribute editor that uses the tag invokes the PULLDOWN element, which formats a field with a drop-down list of values.

This tag takes the ITEM and QUERYASSETNAME: parameters which are the source of the names in the drop-down list. To specify the names, use the ITEM parameter. To specify a query asset that obtains the names dynamically from a database table, use the QUERYASSET parameter.

Note the following:

- You must use a query asset to use a query. You cannot use a SQL statement.

- The SQL in the query asset must return a value column. For example: select name as value from shippingtype.

- If the data type of the attribute using the attribute editor is asset, the query must also return the assets' IDs. For example: select name as value, id as assetid from Product where...

- If you want to include date options, they must be in the format: YYY-MM-DD HH:MM:SS.

The following attribute editor code specifies that the list holds the items red, green, and blue:

```
<?XML VERSION="1.0"?>
<!DOCTYPE PRESENTATIONOBJECT SYSTEM "presentationobject.dtd">

<PRESENTATIONOBJECT NAME="PulldownTest">
<PULLDOWN>
  <ITEM>Red</ITEM>
  <ITEM>Green</ITEM>
  <ITEM>Blue</ITEM>
</PULLDOWN>
```

For sample code that shows how to use the QUERYASSETNAME parameter rather than ITEM, see CHECKBOXES.

Notes about data types:

A PULLDOWN attribute editor is appropriate for attributes with the following data types:

- date

- float

- `integer`

- `money`

- `string`

- `asset`

A drop-down list is the default input style for attributes of type asset. The list displays all the assets of that type. Use a `PULLDOWN` attribute editor when you want to further restrict the items in the drop-down list with a query asset so that the list doesn't display every asset of that type.

9.1.2.6 RADIOBUTTONS

The `presentationobject.dtd` defines a `RADIOBUTTONS` tag. An attribute editor that uses the tag invokes the `RADIOBUTTONS` element, which creates a set of radio options for the attribute.

The `RADIOBUTTONS` tag takes the following parameters:

- `ITEM` or `QUERYASSETNAME`: The source of the names listed next to the radio options. To specify the names, use the `ITEM` parameter. To specify a query asset that obtains the names dynamically from a database table, use the `QUERYASSET` parameter.

  Note the following:

  - You must use a query asset to use a query. You cannot use a SQL statement.

  - The SQL in the query asset must return a value column. For example: `select name as value from shippingtype.`

  - If the data type of the attribute using the attribute editor is `asset`, the query must also return the assets' IDs. For example: `select name as value, id as assetid from Product where...`

- `LAYOUT`: Whether the buttons should be positioned in a vertical list or spread out in a horizontal row. Valid options are `HORIZONTAL` or `VERTICAL`. The default is `HORIZONTAL.`

The following attribute editor code specifies that the `RADIOBUTTONS` element should use the results of a query asset named `A Prods` for the names of a vertical list of buttons:

```
<?XML VERSION="1.0"?>
<!DOCTYPE PRESENTATIONOBJECT SYSTEM "presentationobject.dtd">

<PRESENTATIONOBJECT NAME="RadioButtonTest">
  <RADIOBUTTONS LAYOUT="VERTICAL">
    <QUERYASSETNAME>A Prods</QUERYASSETNAME>
  </RADIOBUTTONS>
</PRESENTATIONOBJECT>
```

For example code that shows the use of the `ITEM` parameter, see PULLDOWN Example.

Notes about data types:

A `RADIONBUTTON` attribute editor is appropriate for attributes with the following data types:

- `date`

- `float`

- `integer`

- `money`

- `string`

- `asset` (If asset, you must supply the name of the query asset that returns the names of the assets.)

> **Note:**
>
> The `RadioButton` attribute editor of type `date` would interpret the date values entered in the `presentation.dtd` in server's timezone (or the timezone set in the JVM). This value will be converted to user's timezone when the asset form is rendered.
>
> To see the same date value irrespective of the timezone, consider using an attribute editor of type `string`.

9.1.2.7 TEXTAREA

The `presentationobject.dtd` defines a `TEXTAREA` tag. An attribute editor that uses the tag invokes the `TEXTAREA` element, which creates a text box field for the attribute, and a pair of radio buttons that allows users to specify whether that attribute should display embedded link buttons.

The `TEXTAREA` tag takes the following parameters:

- `WIDTH`: The text box width, in pixels. If you do not specify a value for `WIDTH`, WebCenter Sites sets a default width. Sample value: `700px`.

- `HEIGHT`: The text box height, in pixels. If you do not specify a value for `HEIGHT`, WebCenter Sites sets a default height. Sample value: `300px`.

> **Note:**
>
> The `WIDTH` and `HEIGHT` parameters replace the deprecated `XSIZE` and `YSIZE` parameters. Setting `XSIZE` and `YSIZE` has no effect.

- `RESIZE`: The `true` value allows users to resize the text area. The default value is `false`.

- `WRAPSTYLE`: Whether the text in the box wraps at all, and, if it does, whether it wraps automatically (soft) or only when the user presses the **Enter** key (a hard return). Valid options are `SOFT`, `HARD`, and `OFF`. The default is `OFF`.

- `DEPTYPE`: The approval dependency between the main asset and an embedded asset. Valid options are `EXISTS` and `EXACT`. The default is `EXACT`.

  - `EXISTS`: When the main asset is edited, approved, and re-published, the embedded asset does not have to be approved and re-published while a version of the asset exists at the destination.

—   EXACT: When the main asset is edited, approved, and re-published, the embedded asset, if it was edited, must be approved and re-published as well.

- MAXVALUES: Limits the number of values that can be added to a multi-valued attribute. For example, if MAXVALUES is set to 10, then only ten values can be added. No default value.

The following attribute editor code defines the XSIZE as 40 pixels, the YSIZE as 5 pixels, disables text wrapping by setting WRAPSTYLE to OFF, and sets the approval dependency for embedded assets to EXISTS:

```
<?XML VERSION="1.0"?>
<!DOCTYPE PRESENTATIONOBJECT SYSTEM "presentationobject.dtd">

<PRESENTATIONOBJECT NAME="TextAreaTest">
  <TEXTAREA XSIZE="40" YSIZE="5" WRAPSTYLE="OFF" DEPTYPE="EXISTS">
  </TEXTAREA>
</PRESENTATIONOBJECT>
```

Notes about data types:

A TEXTAREA attribute editor is appropriate for attributes with the data type of text and blob. Use the text data type when you have to store up to 2000 characters. To store more than 2000 characters, use the blob data type.

9.1.2.8 TEXTFIELD

The presentationobject.dtd defines a TEXTFIELD tag. An attribute editor that uses the tag invokes the TEXTFIELD element from the New and Edit forms to create a text field for the attribute. When the attribute is displayed on the Inspect form, however, it uses the DisplayTEXTFIELD element.

The TEXTFIELD tag takes the following parameters:

- XSIZE: The length of the field, in characters.

- MAXCHARS: The number of characters, up to 256, allowed in the field.

- BLANKED: Whether the attribute's value is replaced with a string of asterisks when it is displayed in the Inspect form. For example, for a password attribute, the value of the password should not be displayed in an Inspect form. Valid options are YES and NO. The default is NO.

    Because using the BLANKED parameter automatically means that you need the field to behave differently on the New and Edit forms than it does on the Inspect form, the TEXTFIELD tag is delivered with both of the two possible elements by default.

- MAXVALUES: Limits the number of values that can be added to a multi-valued attribute. For example, if MAXVALUES is set to 10, then only ten values can be added. No default value.

The following attribute editor code defines the XSIZE as 60 and the maximum number of characters as 80:

```
<?XML VERSION="1.0"?>
<!DOCTYPE PRESENTATIONOBJECT SYSTEM "presentationobject.dtd">

<PRESENTATIONOBJECT NAME="TextFieldTest">
  <TEXTFIELD XSIZE="60" MAXCHARS="80">
```

```
    </TEXTFIELD>
</PRESENTATIONOBJECT>
```

Notes about data types:

A `TEXTFIELD` attribute editor is appropriate for attributes with the following data types:

- `float`

- `integer`

- `money`

- `string`

- `url`

9.1.2.9 TYPEAHEAD

The `presentationobject.dtd` defines a `TYPEAHEAD` tag. An attribute editor that uses the tag invokes the `TYPEAHEAD` element, which creates a type ahead input box with a drop-down menu attribute. When the attribute is displayed in the New and Edit views of an asset, and the down-arrow in the drop-down field is clicked, the drop-down menu lists the names of the assets from which the user can choose.

The `TYPEAHEAD` tag takes the following parameters:

- `PAGESIZE`: Limits the number of results that are shown in the drop-down menu at one time for each type ahead search. For example, if `PAGESIZE` is set to `5`, then only five assets are displayed in the drop-down menu at one time. The default value is 10.

- `MAXVALUES`: Limits the number of values that can be added to a multi-valued attribute. For example, if `MAXVALUES` is set to `10`, then only ten values can be added. No default value.

The `PAGESIZE` parameter can be used to limit the number of results shown in the drop-down at one time. This paginates the results listed in the drop-down menu. For example, if the `PAGESIZE` parameter is set to show 10 assets at a time in the drop-down menu, a user can see the next 10 assets in the list by clicking **More choices**. Similarly, to go back to the last 10 assets that were listed, the user can click the **Previous choices** link.

The list of results can also be filtered by typing the first couple of letters that an asset starts with into the input box. For example, if `Ski` is typed into the input box, the results that are displayed in the drop-down menu (in alphabetic order) will all start with Ski. This is similar to running the query `Ski*`, which finds all assets that begin with the letters or word Ski. As you type into the type ahead input box, it auto-fills with the first result in the drop-down menu, as shown in this figure:

Figure 9-1   Related Content Dialog
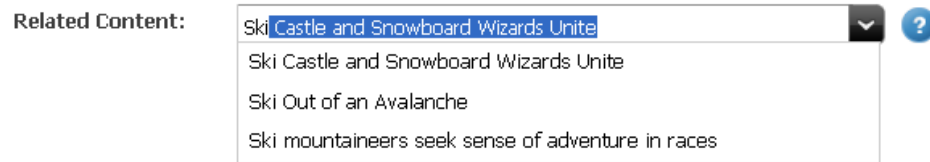
To select the auto-filled asset, users can press **Enter** on their keyboards. Users can also select any result, one at a time, from the drop-down menu, either by selecting an asset (navigating the drop-down menu with the up and down arrow keys on the keyboard) and then pressing the **Enter** key or by using your mouse to click an asset. After you select an asset, it is displayed below the drop-down menu. When a user points to a selected asset, a tooltip is shown, as in this figure:

Figure 9-2   Tooltip Message
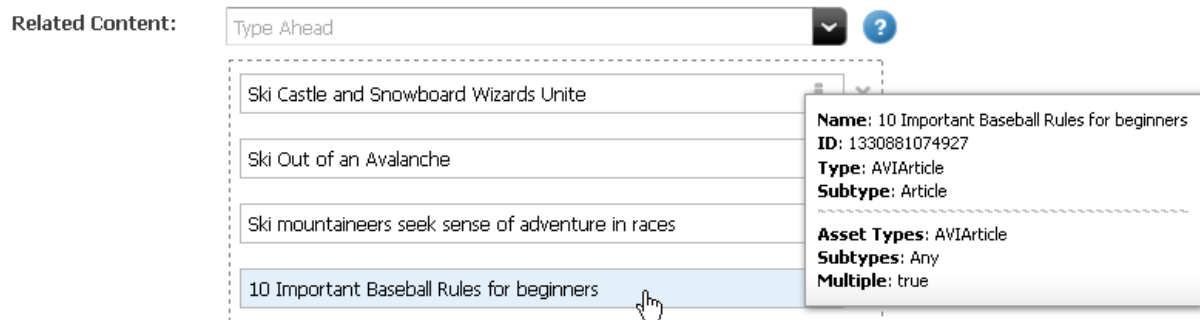
To change the behavior of the TYPEAHEAD attribute editor, for example, to customize the look and feel of the tooltip, see Building an Attribute Editor.

Notes about data types:

A TYPEAHEAD attribute editor is appropriate only for attributes of data type asset.

9.1.2.10 UPLOADER

The presentationobject.dtd defines an UPLOADER tag. An attribute editor that uses the tag invokes the UPLOADER element, which creates an upload attribute. The **Upload** field can upload files of any type. However, to restrict the types of files that can be uploaded, use the FILETYPES parameter.

When a user clicks to upload a file, the progress of the upload is shown in the asset's toolbar, as shown in this figure:

Figure 9-3   Upload Progress
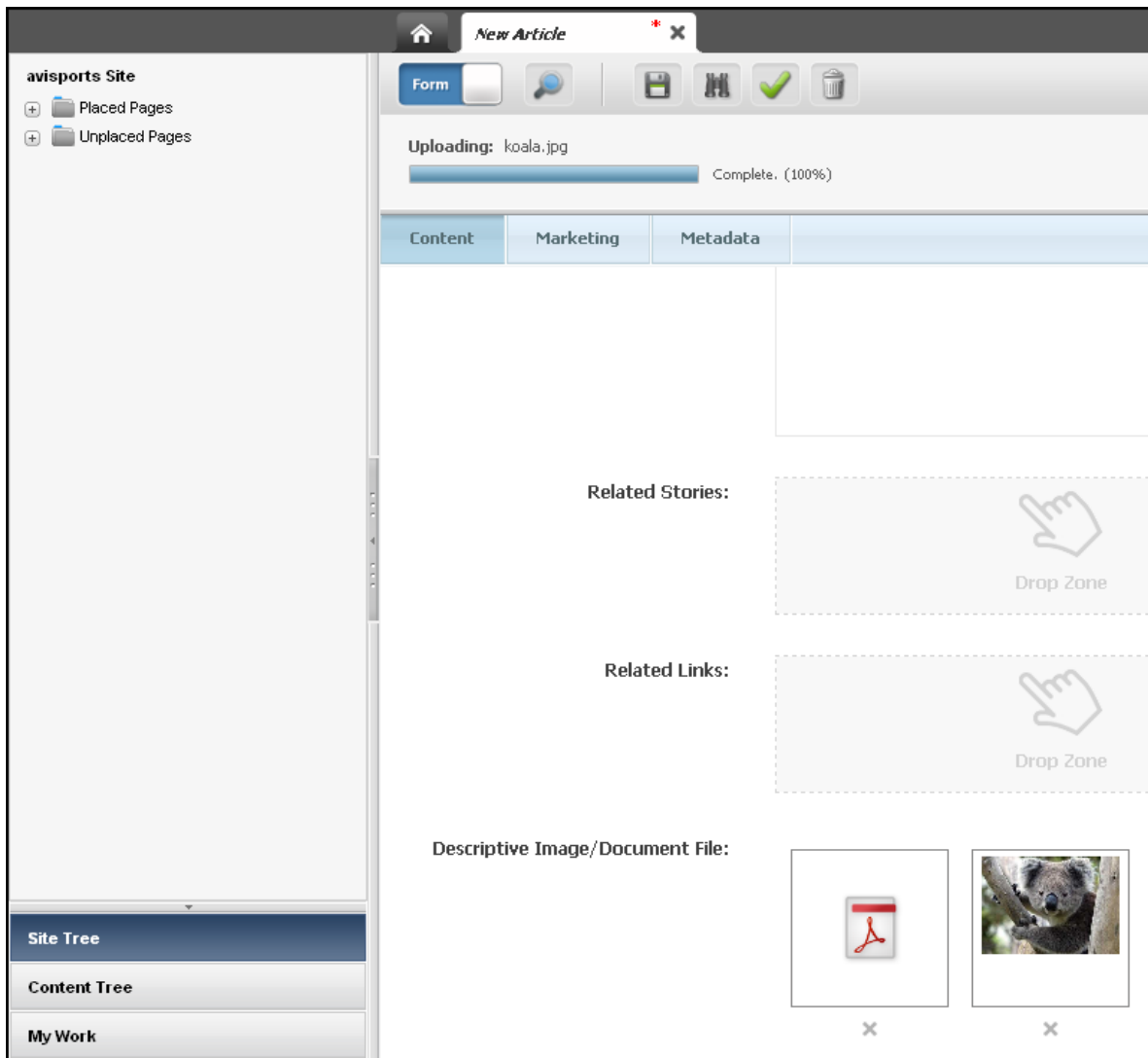
A thumbnail image for each uploaded file is displayed in the **Uploader** field. When a user clicks the thumbnail of an uploaded image file, the image is displayed in an overlay at its actual size. For file types other than images, WebCenter Sites displays a thumbnail image specific to file type in the **Uploader** field. When a thumbnail image for a particular file type is not available, an arbitrary image is shown, instead.

When a user points to the thumbnail image of an image file, the file's name, type, and dimensions are displayed in a tooltip. The file's name and type is displayed in a tooltip of a file that is not an image, as shown in this figure:

Figure 9-4   Image Tooltip
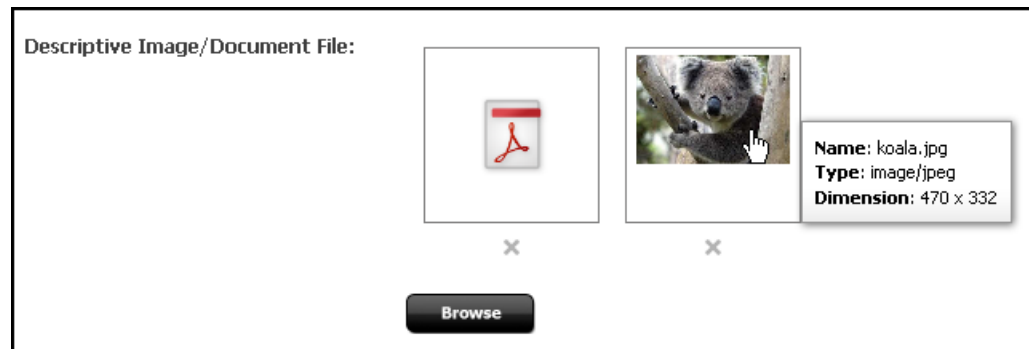
The UPLOADER tag takes the following parameters:

- MAXFILESIZE: Lets you specify the maximum size of the files that can be uploaded. Valid units for this parameter are B, KB, MB, and GB. Default value: 1024KB. Default unit: KB.

- FILETYPES: Lets you specify the types of files that can be uploaded. For example, *.jpg*. If you do not specify a value for this parameter, the default value is any (*.*).

- MINWIDTH: Lets you specify the minimum width (in pixels) of the files that can be uploaded. For example, if MINWIDTH is set to 700px, then users can only upload files with a width that is no less than 700px. If you do not specify a value for this parameter, the default value is any width. Valid unit: px.

- MAXWIDTH: Lets you specify the maximum width (in pixels) of the files that can be uploaded. For example, if MAXWIDTH is set to 700px, then users can only upload files with a width that is no more than 700px. If you do not specify a value for this parameter, the default value is any width. Valid unit: px.

- MINHEIGHT: Lets you specify the minimum height (in pixels) of the files that can be uploaded. For example, if MINHEIGHT is set to 700px, then users can only upload files with a height that is no less than 700px. If you do not specify a value for this parameter, the default value is any height. Valid unit: px.

- MAXHEIGHT: Lets you specify the maximum height (in pixels) of the files that can be uploaded. For example, if MAXHEIGHT is set to 700px, then users can only upload files with a height that is no more than 700px. If you do not specify a value for this parameter, the default value is any height. Valid unit: px.

- MAXVALUES: Limits the number of values that can be added to a multi-valued attribute. For example, if MAXVALUES is set to 10, then only ten values can be added. No default value.

Notes about data types:

A UPLAODER attribute editor is appropriate only for attributes of data type blob.

9.1.3 The Attribute Editor Elements

The elements that take the input values passed to them from their attribute editor counterparts, supply the logic behind the format and behavior of the attribute when it is displayed on a form. For example, it might perform a loop sequence for multivalue attributes so that additional values can be entered in the field.

Table 9-1 lists the default flex attribute display elements located in the `ElementCatalog` table under `OpenMarket/Gator/AttributeTypes`. The names of these elements match exactly the names of the custom XML tags defined in the `presentationobject.dtd` file.

Table 9-1   Attribute Editor Elements

Element	Description
CHECKBOXES	Formats the input style of the attribute as a set of check box options. The attribute editor must either define the names of the options or provide the name of a query asset to use to obtain the names.
CKEDITOR	Invokes the CKEditor text editor. The attribute editor must specify the height and width pixel dimensions for the text box.
PULLDOWN	Formats the input style of the attribute as a select field with a drop-down list. The attribute editor must either specify the items that are displayed in the list or provide the name of a query asset to use to obtain the values.
RADIOBUTTONS	Formats the input style of the attribute as a set of radio options. The attribute editor must define the names of the options or provide the name of a query asset to use to obtain the names.
TEXTAREA	Formats the input style of the attribute as a text box and displays radio buttons that allow the user to specify whether the text box will allow embedded links. The attribute editor must define the x and y dimensions of the box.
TEXTFIELD	Formats the input style of the attribute as a text field. The attribute editor must define the length of the field and the number of characters that are allowed in the field.
DisplayTEXTFIELD	Formats the appearance of the text field attribute's value when it is displayed on the Inspect form. If the attribute editor sets the `BLANKED` parameter to `YES`, this element displays the value from the field as a string of asterisks. Typically used for password fields.
PICKASSET	Formats the input style of the attribute as a **Drop Zone** field into which a user can drag and drop an asset from a tree or search results list.
PICKFROMTREE	Deprecated. Use `PICKASSET`.
TYPEAHEAD	Formats the input style of the attribute as a type ahead input box with a drop-down menu. When the asset is displayed in the New or Edit view of an asset, and the down-arrow in the drop-down field is clicked, the drop-down menu lists the names of the assets from which the user can choose. Use the `PAGESIZE` parameter to limit the number of results shown in the drop-down menu at one time.
UPLOADER	Creates an `upload` attribute. The upload field can upload files of any type. Use the `FILETYPES` parameter to restrict the types of files that can be uploaded.

9.1.4 Conventions for the Attribute Editor Elements

WebCenter Sites can use an element for an attribute editor when that element conforms to the following rules:

- It must have the same name as the input style tag that calls it from the attribute editor code. For example, the default `CHECKBOXES` tag has a default `CHECKBOXES.xml` element.

- The element must be placed in the `ElementCatalog` using the following naming conventions: `OpenMarket/Gator/AttributeTypes/name`.

To create your own display elements for custom attribute editors, find one that is the closest to the attribute editor element that you want to create and then copy as much of it as possible. For help, examine the code in the default attribute editor elements and read the following descriptions of the variables and syntax in them:

Variables:

When WebCenter Sites loads a form that uses the attribute editor, it calls the element with the computer name. It passes the information in the following variables to the display element:

- `PresInst`: The instance of the current presentation object.

- `AttrName`: The name of the current attribute.

- `AttrType`: The data type of the current attribute.

- `EditingStyle`: Whether the attribute can take multiple values (based on the value in the **Number of Values** field for the attribute). This variable is set to either `single` or `multiple`.

- `RequiredAttr`: Whether the attribute is required for the current asset. The variable is set to either `true` or `false`.

- `MultiValueEntry`: Instructs WebCenter Sites how to handle the values for an attribute that can take multiple values.

  When this value is set to `yes`, the display element is called once, under the assumption that the widget created by the element enables the user to select multiple values in it (a multi-select drop-down list, for example).

  When this value is set to `no`, WebCenter Sites calls the display element once for each possible value for the attribute and displays one widget for each value that can be stored.

  Note that this value is always set to `yes` initially.

- `doDefaultDisplay`: Whether to use the default input style for an attribute of this type. For a list, see Default Input Styles for Attributes. When WebCenter Sites calls the display element, this variable is initially set to `yes`. To use the input widget created by the element, the element must reset this variable to `no`.

- `AttrValueList`: The list of all the values for this attribute.

- `TempVal`: The value of a single attribute value.

Other required syntax:

The code in the display element must also use the following conventions:

- It must store information about how to validate the attribute values in a variable named `RequireInfo`. WebCenter Sites passes this variable to elements that use JavaScript to validate the attribute values. Those elements are:

  ```
  OpenMarket/Gator/FlexibleAssets/FlexAssets/ContentForm1
  OpenMarket/Gator/FlexibleAssets/FlexGroups/ContentForm1
  ```

  This JavaScript performs prescribed error checking and validation based on the type of control, the data type, and other predictable characteristics. The information passed in the `RequireInfo` variable informs the JavaScript about the custom requirements for the attribute editor.

- The name of the widget in the display element (the `INPUT NAME`) must use the following convention:

  - For a single-value attribute, the name of the attribute.

  - For a multi-value attribute, it must use a 1-based counter prepend the attribute name for each attribute value (for example, 1color, 2color, 3color).

For an example, see Customizing Attribute Editors.

9.2 Creating Attribute Editors

To give you a hands-on experience of creating an attribute editor, the sample XML is provided in the Samples subdirectory. If you're creating an attribute editor that you'll add to a flex or flex parent asset's form of your online site, write your own XML code or edit the sample code suitably.

1. Open the Admin interface.

2. Click **New** and then select **New Attribute Editor** from the shortcut list.

   The Attribute Editor form opens.

Figure 9-5   Attribute Editor Dialog

3. In the **Name** field, enter a unique name of up to 64 characters, excluding spaces.

4. In the **Description** field, enter a short phrase that describes the purpose of the attribute editor.

5. In the **XML** field, either paste the appropriate sample XML attribute editor code from the HTML version of this guide or from the sample text files provided in the `Samples` subdirectory of the installation directory.

6. Edit the code as needed. For example, for a CHECKBOXES or a RADIOBUTTONS attribute editor, you must provide names for the check boxes or radio buttons. If you are creating a PULLDOWN attribute editor, you must provide the values for the drop-down list.

7. Click **Save**.

> **Note:**
>
> Another option is to code the XML for the attribute editor in a separate .xml file. In this case, rather than entering the code directly into the **XML** field, click **Browse** next to the **XML in file** field and select the file.

8. Approve this attribute editor so that it can be published to the management system.

> **Note:**
>
> If you are using a query asset with this attribute editor, be sure to approve both the attribute editor and the query asset.
>
> Because the dependency between an attribute editor and its query asset is specified in the XML code in the attribute editor, the approval system cannot detect the dependency and verify that the query asset exists on the management system.

9.3 Customizing Attribute Editors

Your custom attribute editor may be similar to the sample editor in many ways. So, you can modify the sample code and display elements to use them for your custom editor. The presentationobject.dtd file needs a new PRESENTATIONOBJECT tag for a new input style. You can't use any of the default tags.

When you create a custom `PRESENTATIONOBJECT` tag, you must also supply the appropriate display elements for it:

- Required: An element that formats the attribute (displays an edit mechanism) when that attribute displays in a New or Edit form.

- Optional: An element that formats the attribute when it displays in the Inspect form.

- Optional: An element that formats the attribute data before it is saved in the database.

For information about the variables and conventions used in the display elements for an attribute editor, see The Attribute Editor Elements.

To create or customize an attribute editor into which a file can be uploaded (for example, if you are customizing the `UPLOADER` attribute editor), add custom logic to the attribute editor code which will ensure that the file being uploaded is valid. See Adding Custom Logic to Validate an Uploaded File.

9.3.1 Example: Customized Attribute Editor

This example demonstrates how to customize the description of the `TEXTAREA` tag in the `presentationobject.dtd` file and the `TEXTAREA` element to create an attribute editor that disables a text box for the user who does not have the proper permissions.

There are three steps:

- Step 1: Editing the description of the `TEXTAREA` tag in the `presentationobject.dtd` to support a new parameter named `PERMISSIONS`.

- Step 2: Writing the code for the attribute editor and creating the attribute editor.

- Step 3: Editing the `TEXTAREA` element to check the value of `PERMISSIONS`.

9.3.1.1 Step 1: Editing the presentationobject.dtd File

To support the new parameter, you add a single line of code to the `TEXTAREA` description in the `presentationobject.dtd`.

- Add the new line added to the end of the default description of the `TEXTAREA` tag, as the following code shows:

```
<!-- TEXTAREA: A text area of a specific size. You must specify
-- the x and y dimensions; the wrap style defaults to soft.
-->
<!ELEMENT TEXTAREA ANY>
<!ATTLIST TEXTAREA XSIZE CDATA #REQUIRED>
<!ATTLIST TEXTAREA YSIZE CDATA #REQUIRED>
<!ATTLIST TEXTAREA WRAPSTYLE (OFF | SOFT | HARD) "SOFT">
<!ATTLIST TEXTAREA PERMISSION CDATA>
```

9.3.1.2 Step 2: Specifying Permission for the Example Attribute Editor

Here is the example code with the new parameter.

- Specify that a user must have Administrators as the value for PERMISSION to see the field, as the following code shows:

```
<?XML VERSION="1.0"?>
<!DOCTYPE PRESENTATIONOBJECT SYSTEM "presentationobject.dtd">


<PRESENTATIONOBJECT NAME="TextAreaTest">
  <TEXTAREA XSIZE="40" YSIZE="10" WRAPSTYLE="SOFT" PERMISSION="Administrators">
  </TEXTAREA>
</PRESENTATIONOBJECT>
```

9.3.1.3 Step 3: Editing the TEXTAREA Element

The third step is editing the TEXTAREA element.

- Add the lines shown in bold in the following code to enable or disable the field, based on the value of the PERMISSION parameter:

```
<?XML VERSION="1.0" ?>
<!DOCTYPE FTCS SYSTEM "futuretense_cs.dtd">
<FTCS Version="1.1">
<!-- OpenMarket/Gator/AttributeTypes/TEXTAREA
--
-- INPUT
--
-- OUTPUT
--
-->

<!-- Display one TEXTAREA per attribute value -->
<IF COND="Variables.MultiValueEntry=no">
<THEN>

<!-- Don't want default display field  -->
<setvar NAME="doDefaultDisplay" VALUE="no"/>

<!-- Get all parameters from Attribute Editor xml -->
<presentation.getprimaryattributevalue
NAME="Variables.PresInst"
ATTRIBUTE="FONTSIZE" VARNAME="FONTSIZE"/>
   <if COND="Variables.errno!=0">
<then>
<setvar NAME="FONTSIZE" VALUE="2"/>
</then>
</if>

<presentation.getprimaryattributevalue
NAME="Variables.PresInst"
ATTRIBUTE="WRAPSTYLE" VARNAME="WRAPSTYLE"/>
```

```
<if COND="IsVariable.WRAPSTYLE!=true">
<then>
<setvar NAME="WRAPSTYLE" VALUE="OFF"/>
</then>
</if>

<presentation.getprimaryattributevalue
NAME="Variables.PresInst"
ATTRIBUTE="XSIZE" VARNAME="XSIZE"/>
<if COND="IsVariable.XSIZE!=true">
<then>
<setvar NAME="XSIZE" VALUE="24"/>
</then>
</if>

<presentation.getprimaryattributevalue
NAME="Variables.PresInst"
ATTRIBUTE="YSIZE" VARNAME="YSIZE"/>
<if COND="IsVariable.YSIZE!=true">
<then>
<setvar NAME="YSIZE" VALUE="20"/>
</then>
</if>

<setvar NAME="disableTextArea" VALUE="no"/>
<presentation.getprimaryattributevalue
NAME="Variables.PresInst"
ATTRIBUTE="PERMISSION" VARNAME="PERMISSION"/>
<if COND="IsVariable.PERMISSION=true">
<then>
<setvar NAME="errno" VALUE="0"/>
<USERISMEMBER GROUP="Variables.PERMISSION"/>
<IF COND="Variables.errno=0">
<THEN>
<setvar NAME="disableTextArea" VALUE="yes"/>
</THEN>
</IF>
</then>
</if>

<tr>

<!-- Standard element to display attribute name or description
-->
<callelement NAME="OpenMarket/Gator/FlexibleAssets/Common
/DisplayAttributeName"/>
<td></td>
<td>

<!-- Single valued attributes -->
<if COND="Variables.EditingStyle=single">
<then>

<!-- Special case: TEXTAREA for URL attributes -->
<IF COND="Variables.AttrType=url">
<THEN>
<setvar NAME="errno" VALUE="0"/>
<BEGINS STR="AttrValueList.urlvalue"
WHAT="AttrValueList."/>
<IF COND="Variables.errno=1">
<THEN>
```

```
<setvar NAME="filename" VALUE="CS.UniqueID.txt"/>
</THEN>
<ELSE>
<setvar NAME="filename"
VALUE="AttrValueList.urlvalue"/>
</ELSE>
</IF>

<INPUT TYPE="hidden" NAME="Variables.AttrName_file"
VALUE="Variables.filename"
REPLACEALL="Variables.AttrName,Variables.filename"/>

<setvar NAME="errno" VALUE="0"/>
<BEGINS STR="AttrValueList.@urlvalue"
WHAT="AttrValueList."/>
<IF COND="Variables.errno=1">
<THEN>
<setvar NAME="MyAttrVal" VALUE="Variables.empty"/>
</THEN>
<ELSE>
<setvar NAME="MyAttrVal"
VALUE="AttrValueList.@urlvalue"/>
</ELSE>
</IF>
</THEN>
</IF>

<!-- Display a TEXTAREA with all parameters from Attribute
--Editor xml -->
<!-- The NAME of the input must be the attribute name -->
<IF COND="Variables.disableTextArea=yes">
<THEN>
<TEXTAREA DISABLED="yes" NAME="Variables.AttrName"
ROWS="Variables.YSIZE" COLS="Variables.XSIZE"
WRAP="Variables.WRAPSTYLE"
REPLACEALL="Variables.AttrName,Variables.XSIZE,
Variables.YSIZE,Variables.WRAPSTYLE,Variables.empty">

<!-- For most single valued attrs, the value is contained in
MyAttrVal -->
<csvar NAME="Variables.MyAttrVal"/>
</TEXTAREA>
</THEN>
<ELSE>
<TEXTAREA NAME="Variables.AttrName"
ROWS="Variables.YSIZE" COLS="Variables.XSIZE"
WRAP="Variables.WRAPSTYLE"
REPLACEALL="Variables.AttrName,Variables.XSIZE,
Variables.YSIZE,Variables.WRAPSTYLE,Variables.empty">
<!-- For most single valued attrs, the value is
contained in MyAttrVal -->
<csvar NAME="Variables.MyAttrVal"/>
</TEXTAREA>
</ELSE>
</IF>
</then>
<else>
<!-- Multiple valued attributes -->
<!-- For single value attributes we can usually use the
default RequireInfo -->
<!-- For multiple value attributes we need to append to
```

```
RequireInfo for each value -->
<if COND="Variables.RequiredAttr=true">
<then>
<setvar NAME="RequireInfo"
VALUE="Variables.RequireInfo*Counters.TCounterVariables.
AttrName*ReqTrue*Variables.AttrType!"/>
</then>
<else>
<setvar NAME="RequireInfo"
VALUE="Variables.RequireInfo*Counters.TCounterVariables
.AttrName*ReqFalse*Variables.AttrType!"/>
</else>
</if>

<!-- Display a TEXTAREA with all parameters from Attribute
Editor xml -->
<!-- The NAME of the input must be the attribute name
prepended by the TCounter counter -->
<IF COND="Variables.disableTextArea=yes">
<THEN>
<TEXTAREA DISABLED ="yes" NAME="Counters.TCounterVariables.AttrName"
ROWS="Variables.YSIZE" COLS="Variables.XSIZE"
WRAP="Variables.WRAPSTYLE"
REPLACEALL="Counters.TCounter,
Variables.AttrName,Variables.XSIZE,
Variables.YSIZE,Variables.WRAPSTYLE">
<csvar NAME="Variables.tempval"/> </TEXTAREA>
</THEN>
<ELSE>
<TEXTAREA NAME="Counters.TCounterVariables.AttrName"
ROWS="Variables.YSIZE" COLS="Variables.XSIZE"
WRAP="Variables.WRAPSTYLE"
REPLACEALL="Counters.TCounter,
Variables.AttrName,Variables.XSIZE,
Variables.YSIZE,Variables.WRAPSTYLE">
<csvar NAME="Variables.tempval"/> </TEXTAREA>
</ELSE>
</IF>
</else>
</if>
</td>
</tr>
</THEN>
</IF> <!-- MultiValueEntry -->
</FTCS>
```

9.3.2 Adding Custom Logic to Validate an Uploaded File

To validate an uploaded file, you add custom logic to an attribute editor's code.

1. Create a JSP element with the name `ValidateFileUpload` in the path `CustomElements/fatwire/ui/util`.

2. In the JSP element, retrieve the file name and the byte array containing the uploaded file data in the request scope by calling `ics.GetCgi("fileBytes")` which returns an FTVAL, from which you can call the `getBlob()` method to get the `byte[]` for the uploaded blob.

```
byte[] fileBytes = null;
if(ics.GetCgi("fileBytes") != null){
```

```
fileBytes = ics.GetCgi("fileBytes").getBlob();
}
```

3. (Optional) Write your custom logic to ensure that the uploaded file data is valid:

- If the file data is valid, set the variable `fileValidated` to `true` by calling `ics.SetVar("fileValidated","true")`. This ensures that the file will be uploaded to the WebCenter Sites server.

- If the file data is invalid, set the `fileValidated` variable to `false` by calling `ics.SetVar("fileValidated","false")`. Additionally, set the custom validation failure message in the `fileUploadFailureMessage` variable by calling `ics.SetVar("fileUploadFailureMessage",<CustomErrorMessage>)` in custom element. This rejects the data uploaded by the user and displays the custom error message adjacent to progress bar for the file being uploaded. If the `fileUploadFailureMessage` variable is not set but `fileValidated` is set to `false`, WebCenter Sites shows a generic message `Upload Failed: <fileName>`.

Note:

Sample code (except validation logic) is available in the base element `fatwire/ui/util/ValidateFileUpload`. This element should not be modified and the custom code must be written in the `CustomElements` path as specified in step 1.

9.4 Considerations About Editing Attribute Editors

Some changes to the attribute editors don't cause data loss, while others may affect the data.

- You can change the **Name** without causing a schema change.

- You can change the **Description** without causing data loss.

- If you change code in the attribute editor:

  – You can add input options.

  – If you have existing data, you should not remove input options. If you do, some of your existing data will no longer be valid and you will have to search through the database and fix it.

  – If you change the input style, you risk a data mismatch.

10

Configuring Bundled Attribute Editors

Image Picker, CKEditor, and Clarkii Online Image Editor come packaged with WebCenter Sites. Content contributors use Image Picker to visually choose an image asset to associate with the asset they are creating or editing. With CKEditor, they apply a wide range of MS Word-style formatting to their content. And, they use the Clarkii Online Image Editor (Clarkii OIE) to edit an image directly in an asset's New or Edit view in Form Mode.

See these topics on configuring instances of attribute editors that ship with WebCenter Sites:

- Configuring CKEditor

- Configuring the Clarkii Online Image Editor

- Configuring the Image Picker

10.1 Configuring CKEditor

CKEditor is an open source WYSIWYG text editor from CKSource which requires no client-side installation. CKEditor is bundled with WebCenter Sites. You can configure asset types containing WYSIWYG-enabled text fields to use CKEditor by default.

See these topics:

- Before You Begin

- How to Create a CKEditor Instance and Enable It for a Field

- How to Enable CKEditor for Use in Web Mode

- How to Enable Selected Asset Types for the CKEditor

- How to Set the Approval Dependency for Included Assets

- How to Customize the CKEditor Toolbar

- How to Configure Spell Check Support in CKEditor

10.1.1 Before You Begin

If you are using CKEditor with Internet Explorer, and WebCenter Sites is configured to use the cp1252 character set, follow these steps to have CKEditor work correctly in Internet Explorer:

1. In the Admin interface, open the Property Management Tool and note the value of the `cs.contenttype` property.

2. Log in to Explorer.

3. Go to the element `SiteCatalog\OpenMarket\xcelerate\Actions` `\FCKEditorRenderer` and set its `resargs1` field to:

```
cs.contenttype=[your content type]; charset=windows-1252
```

where `[your content type]` takes the value that you noted from the Property Management Tool.

10.1.2 How to Create a CKEditor Instance and Enable It for a Field

This procedure shows you how to create an instance of CKEditor and enable it for a flex asset attribute based on a FirstSiteII example.

In our example, you will enable a new CKEditor instance as the input method (attribute editor) for the **FSIIAbstract** field. (FSIIAbstract is a flex attribute of the `Content` sample asset type.)

To Create and Enable a CKEditor Instance:

1. Open the Admin interface.

2. Create the CKEditor instance:

   a. In the button bar, click **New**.

   b. In the list of asset types, click **New Attribute Editor**.

   WebCenter Sites displays the New Attribute Editor form.

   c. In the **Name** field, enter a name that uniquely identifies this instance of CKEditor. For the purpose of our example, enter **CK_FSIIAbstract**.

   Clients can use generic names to create the CKEditor and use it for multiple attributes. It is not required to uniquely identify the CKEditor instance for the attribute unless there is a specific requirement for that field. For example, the width and height of the CKEditor in that field if it is different than other CKEditor fields.

   d. Paste the following code into the XML field:

   ```
   <?XML VERSION="1.0"?>
   <!DOCTYPE PRESENTATIONOBJECT SYSTEM "presentationobject.dtd">
   <PRESENTATIONOBJECT NAME="CKEDITOR">
   <CKEDITOR WIDTH="400" HEIGHT="200"></CKEDITOR>
   </PRESENTATIONOBJECT>
   ```

   e. Click **Save**.

3. Enable the CKEditor instance as the input method for the **FSIIAbstract** field. Find and open the `FSIIAbstract` attribute asset in the Edit form:

   a. In the button bar, click **Search**.

   b. In the list of asset types, click **Find Content Attribute**.

   c. In the **Search** field, enter `FSIIAbstract` and click **Search**.

   d. In the list of search results, navigate to the **FSIIAbstract** asset and click its **Edit** (pencil) icon.

   WebCenter Sites opens the asset in the Edit form.

      e. Set CKEditor as the attribute editor (input method) for this attribute. In the Attribute Editor drop-down list, choose **CK_FSIIAbstract**.

      f. Click **Save**.

4. To test your new CKEditor instance, switch to the Oracle WebCenter Sites: Contributor interface, and find and open any Content asset in its Edit view:

      a. In the **Search** field, click the down-arrow icon to open the **Search Type** menu. In the **Search Type** menu, choose **Find Content**. Then click the **magnifying glass** icon.

      A Search tab opens displaying the results of your search.

      b. In the list of asset types, navigate to a `Content` asset of your choice (**FSIIAbout** in our example), right-click the asset and choose **Edit** from the context menu.

      A tab opens displaying the asset in its Edit view.

      c. Navigate to the **Abstract** field and click in the field.

      If CKEditor does not appear, check the attribute editor XML code and the selection you made in the **Attribute Editor** drop-down list.

      d. In the asset's toolbar, click the **Inspect** icon to display the asset's Inspect view.

10.1.3 How to Enable CKEditor for Use in Web Mode

To enable users to edit assets using CKEditor in Web Mode, set the `editor` and `params` parameters in the `insite:edit` tag within the appropriate template, as shown in the table below. The template whose code you edit must be either for the asset whose CKEditor you want to work with in Web Mode, or for the asset type associated with that asset.

Table 10-1   Parameters that Enable CKEditor in Web Mode

Tag	Parameter	Value	Description
`insite:edit`	`editor`	`ckeditor`	Specifies the name of the editor to use.
N/A	`params`	`editorId`	Specifies the ID of the CKEditor you want to use in Web Mode.
N/A	N/A	`enableEmbedd edLinks`	Enables the link icons on the CKEditor toolbar.

This example shows the required parameters for enabling CKEditor in Web Mode:

Example 10-1   Code to Enable CKEditor in Web Mode

```
<ics:listget listname='BodyList' fieldname="value" output="Body" />
<div id="body">
<insite:edit
     assetid='<%=ics.GetVar("cid")%>'
     assetfield='<%='Attribute_"+ics.GetVar("BodyAttrName")%>'
     assetfieldvalue='<%=ics.GetVar("Body")%>'
     assettype='<%=ics.GetVar("c")%>'
     editor='ckeditor'
```

```
        params="{enableEmbeddedLinks:'1'}"/>
</div>
```

> **Note:**
>
> If the insite:edit tag does not define a height and width for CKEditor, then CKEditor will use the XSIZE and YSIZE defined in its XML code. See the *Oracle Fusion Middleware Tag Reference for Oracle WebCenter Sites Reference*.

You can edit the insite:edit tag in one of the following ways:

- In the **Element Logic** field of the appropriate template. To work in a template's **Element Logic** field:

    – Open the Admin interface.

    – Click **Search** (in the button bar), then select **Find Template**, then **Search**, and then *Select template name.*

    – From the template's Edit form, select the **Element** tab and navigate to the **Element Logic** field.

- In Oracle WebCenter Sites Explorer.

- In a text editor of your choice.

10.1.4 How to Enable Selected Asset Types for the CKEditor

To narrow down the user's choice of available asset types, add the ALLOWEDASSETTYPES parameter to the CKEditor XML code. As values, specify the names of the allowed asset types in a comma-separated list.

To enable asset types for the CKEditor:

1. Open the Admin interface.

2. Find and open the attribute editor named CKEditor in its Edit form:

    a. In the button bar, click **Search**.

    b. In the Search form, select **Find Attribute Editor**.

    c. In the Search for Attribute Editors form, fill in a search criteria (if any) and click **Search**.

    d. In the search results list, navigate to the **CKEditor** asset and click its **Edit** (pencil) icon.

3. Navigate to the **XML** field, add the ALLOWEDASSETTYPES parameter to the <CKEDITOR> tag. Then, specify the names of the asset types you want to enable in the value of the ALLOWEDASSETTYPES parameter, as shown in this example:

```
<?XML VERSION="1.0"?>
<!DOCTYPE PRESENTATIONOBJECT
SYSTEM "presentationobject.dtd">
<PRESENTATIONOBJECT
NAME="CKEDITORTEST"><CKEDITOR WIDGET="580"
HEIGHT="200" IMAGEPICKERID="1112668339899"
```

```
ALLOWEDASSETTYPES="MEDIA_C,CONTENT_C,PRODUCT_C,DOCUMENT_C">
</CKEDITOR></PRESENTATIONOBJECT>
```

The ALLOWEDASSETTYPES parameter is added as an additional parameter to the <CKEDITOR> tag. The value of this parameter specifies the asset types with which the user can create a new asset.

4. Click **Save Changes** to save the asset.

5. Test the ALLOWEDASSETTYPES parameter:

   a. Switch to the Contributor interface.

   b. Find and open an asset with a CKEditor enabled field in Web Mode:

      • In the **Search** field, specify a search criteria and then click the **magnifying glass** icon.

        A Search tab opens displaying the results of your search.

      • Click the name of an asset to open its Inspect view.

      • If the asset opens in Form Mode, click the **Mode** switch in the asset's toolbar to switch to Web Mode.

      • In the asset's toolbar, click the **Edit** icon.

   c. Search on one of the allowed asset types. After the allowed asset displays in the docked search results list, drag the allowed asset to CKEditor.

      If you do not have any allowed assets of the type you specified with the ALLOWEDASSETTYPES parameter, select **Content** and then **New** to create a new asset. Then search on the asset and drag it from the dock to the CKEditor.

10.1.5 How to Set the Approval Dependency for Included Assets

The **Include asset** and **Create and include a new asset** icons allow users to include one asset in another asset's CKEditor-enabled field. The included asset is then previewable in the field and, ultimately, embedded in the display of the main asset online.

After the main and included assets have been published for the first time, the dependency between them determines how subsequent approvals and publications will work. This dependency is defined by the DEPTYPE parameter in the XML code of the CKEditor.

The DEPTYPE parameter can be set to either EXISTS or EXACT. If the DEPTYPE parameter is not set explicitly, EXACT is used by default.

• EXISTS: When the main asset is edited, approved, and re-published, the included asset does not have to be approved and re-published while a version of the asset exists at the destination.

• EXACT: When the main asset is edited, approved, and re-published, the included asset, if it was edited, must be approved and re-published as well.

To Define the Approval Dependency for Included Assets:

1. Open the Admin interface.

2. Find and open the attribute editor named CKEditor in its Edit form:

   a. In the button bar, click **Search**.

   b. In the Search form, select **Find Attribute Editor**.

   c. In the Search for Attribute Editors form, fill in a search criteria (if any) and click **Search**.

   d. In the search results list, navigate to the **CKEditor** asset and click its **Edit** (pencil) icon.

3. Navigate to the **XML** field, add the DEPTYPE parameter to the <CKEDITOR> tag, and set the value of the DEPTYPE parameter to either EXISTS or EXACT, as shown in this example:

```
<?XML VERSION="1.0"?>
<!DOCTYPE PRESENTATIONOBJECT
SYSTEM "presentationobject.dtd">
<PRESENTATIONOBJECT
NAME="CKEditorTest"><CKEDITOR
WIDTH="580" HEIGHT="200"
DEPTYPE="EXISTS"></CKEDITOR>
</PRESENTATIONOBJECT>
```

The DEPTYPE parameter is added as an additional attribute to the <CKEDITOR> tag.

4. Click **Save Changes** to save the asset.

10.1.6 How to Customize the CKEditor Toolbar

You can customize the functions available in the CKEditor toolbar and their arrangements in the custom configuration file.

To customize the CKEditor toolbar in the custom config.js file:

1. Open the configuration file in a text editor:

   <cs_app>/ckeditor/customconfig.js

   where <cs_app> refers to the directory into which the WebCenter Sites web application was deployed on your application server.

2. To customize the toolbar for the Contributor interface (for Form Mode and Web Mode):

   • locate config.toolbar_SITES in <cs_app>/ckeditor/customconfig.js for customizing the toolbar for the Contributor interface.

   • locate config.toolbar_SITES_WEB in <cs_app>/ckeditor/customconfig.js for customizing the toolbar for Web Mode.

3. Make your changes. For information about the toolbar definition syntax, consult the CKEditor documentation.

4. Save and close the file.

5. Redeploy the WebCenter Sites application and restart the application server for your changes to take effect.

To customize the CKEditor toolbar on a per-instance basis:

1. Create the custom toolbar definition:

   a. Open the custom configuration file in a text editor:

      ```
      <cs_app>/ckeditor/config.js
      ```

      where `<cs_app>` refers to the directory into which the WebCenter Sites web application was deployed on your application server.

   b. Add a new toolbar definition to the file. For information about how to build a custom toolbar definition, consult the CKEditor documentation.

      ```
      config.toolbar_<toolbardef> = [
                          [ 'Cut', 'Copy', 'Paste', 'PasteText', 'PasteFromWord',
      '-', 'Undo', 'Redo' ],
                          { name: 'links', items: [ 'Link', 'Unlink',
      'Anchor' ] },
                          { name: 'basicstyles', items: [ 'Bold', 'Italic',
      'Underline','Strike','-','Subscript','Superscript' ] },
                          { name: 'colors', items: [ 'TextColor', 'BGColor' ] },
                          { name: 'tools', items: [ 'Maximize', 'ShowBlocks' ] }
          ];
      ```

      The `<toolbarDef>` value is the name of your custom toolbar definition. You will use this name when modifying a CKEditor instance to use this custom definition.

2. Modify a **CKEditor** instance to use your custom toolbar definition.

   a. Open the Admin interface.

   b. Find the CKEditor instance:

      - In the button bar, click **Search**.

      - In the list of asset types, click **Find Attribute Editor**.

      - In the **Search** field, enter the name of the asset holding the CKEditor instance and click **Search**.

      - In the list of search results, navigate to the asset and click its **Edit** (pencil) icon.

   c. In the **XML** field, modify the attribute editor code as follows:

      - Find the line highlighted in bold below:

        ```
        <?XML VERSION="1.0"?>
        <!DOCTYPE PRESENTATIONOBJECT SYSTEM
            "presentationobject.dtd">
        <PRESENTATIONOBJECT NAME="CKEditorCustomized">
        <CKEDITOR WIDTH="400" HEIGHT="200">
        </CKEDITOR></PRESENTATIONOBJECT>
        ```

      - Add the `TOOLBAR` parameter to the `<CKEDITOR>` tag:

        ```
        TOOLBAR="<toolbarDef>"
        ```

      - The value of the `TOOLBAR` parameter specifies the name of the custom toolbar definition you created in step 1.

- The modified line should look as follows:

```
<CKEDITOR WIDTH="400" HEIGHT="200" TOOLBAR="<toolbarDef>">
```

    d. Click **Save Changes** to save the asset.

3. Redeploy the WebCenter Sites application and restart the application server for your changes to take effect.

For information about configuring CKEditor, consult its documentation.

10.1.7 How to Configure Spell Check Support in CKEditor

For instructions about using and configuring the CKEditor spell checker, visit the following URL: http://docs.cksource.com/CKEditor_3.x/Users_Guide/Spell_Checking

10.2 Configuring the Clarkii Online Image Editor

The Clarkii Online Image Editor (Clarkii OIE) is a third-party image editor from InDis Baltic that is supported on all browsers on which WebCenter Sites is supported, including Safari. You can configure separate instances of Clarkii OIE on a per-field basis for each asset type, or a single instance to be associated with the fields of multiple asset types.

The following figure summarizes the native controls of Clarkii OIE and functions provided by WebCenter Sites for operating on images in an attribute for which Clarkii OIE is enabled.

Figure 10-1   Clarkii Online Image Editor Rendered in a Field of an Asset's Form
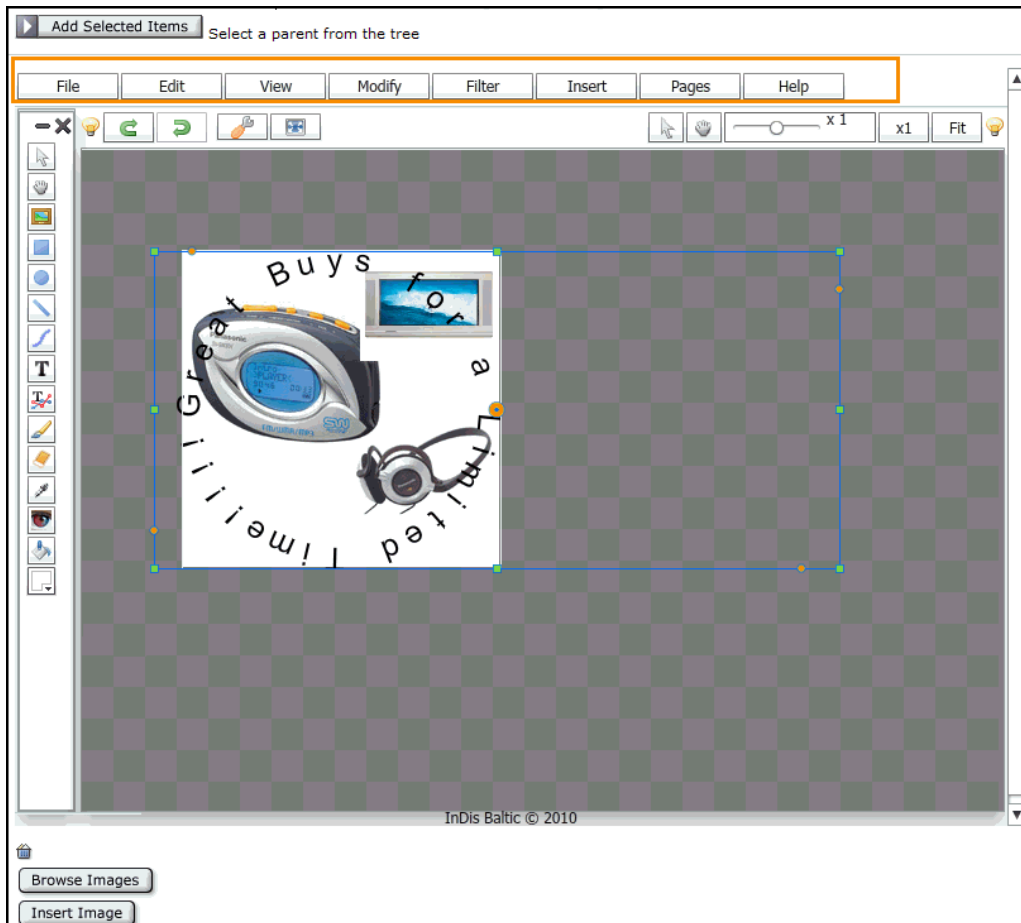
Before configuring and enabling this feature, take note of the following:

- Clarkii OIE can be enabled only for flex attributes. The instructions in this section use the Media flex family of the FirstSiteII sample site as an example.

- Clarkii OIE can be enabled only for attributes of type `blob`.

- Flash must be installed on the client browser in order for Clarkii OIE to be rendered in the field for which you enabled it.

Note:

You can customize the functions in the Clarkii OIE toolbar and menu. Since these functions are strictly Clarkii OIE related they are not documented. For instructions about configuring Clarkii OIE specific functions, visit the following URL:

`http://www.online-image-editor-clarkii.com/`

Topics:

- How to Create a Clarkii OIE Instance and Enable it for a Field

- How to Configure Clarkii OIE Properties

- How to Implement a Field Copier Filter to Classify Assets

10.2.1 How to Create a Clarkii OIE Instance and Enable it for a Field

Create a Clarkii OIE instance and enable it for a flex attribute asset. This procedure is based on the FirstSiteII sample site.

In this example, you will enable a new Clarkii OIE instance as the attribute editor for the **FSII_ImageFile** field. FSII_ImageFile is an attribute of the sample Media asset type.

1. Open the Admin interface.

2. Create a Clarkii OIE instance:

    a. In the button bar, click **New**.

    b. In the list of asset types, click **New Attribute Editor**.

        WebCenter Sites displays the New Attribute Editor form.

    c. In the **Name** field, enter a name that uniquely identifies this Clarkii OIE instance (for this example, enter ClarkiiOIE).

    d. In the **XML** field, paste the following code.

    > **Note:**
    >
    > - For detailed information about each parameter, see the table in How to Configure Clarkii OIE Properties.
    >
    > - The attribute editor must include Buttons such as those included in the XML code below. Otherwise a blank page is displayed in the Edit form.

```
<?XML VERSION="1.0"?>
<!DOCTYPE PRESENTATIONOBJECT SYSTEM "presentationobject.dtd">
<PRESENTATIONOBJECT NAME="editor">
<IMAGEEDITOR
  EDITORTYPE="clarkii"
  HEIGHT="500"
  WIDTH="600"
  MAXHEIGHT="650"
  MAXWIDTH="900"
  FITIMAGE="true"
  ENABLEOIEFORMAT="true"
  DEFAULTTEXTFONT="Arial"
  DEFAULTTEXTSIZE="18"
  DEFAULTTEXTCOLOR="#000000"
  ENABLEIMAGEPICKER="true"
  ASSETTYPE="Media_C"
  ATTRIBUTE="FSII_ImageFile"
  ATTRIBUTETYPE="Media_A"
  CATEGORYATTRIBUTE=""
  RESTRICTEDCATEGORYLIST=""
  OIEASSETTYPE="Media_C"
  OIEATTRIBUTE="FSII_ImageFile"
  OIEATTRIBUTETYPE="Media_A"
  OIECATEGORYATTRIBUTE=""
  OIERESTRICTEDCATEGORYLIST=""
```

```
                       OIEENABLEIMAGEPICKER="true"
                       TAGEDIT="true">
                       <BUTTONS>
                         <BUTTON NAME="New" VISIBLE="true"/>
                         <BUTTON NAME="Open" VISIBLE="true"/>
                         <BUTTON NAME="Scan" VISIBLE="false"/>
                         <BUTTON NAME="Save" VISIBLE="true"/>
                         <BUTTON NAME="Copy" VISIBLE="true"/>
                         <BUTTON NAME="Paste" VISIBLE="true"/>
                         <BUTTON NAME="Undo" VISIBLE="true"/>
                         <BUTTON NAME="Redo" VISIBLE="true"/>
                         <BUTTON NAME="Brush" VISIBLE="true"/>
                         <BUTTON NAME="Eraser" VISIBLE="true"/>
                         <BUTTON NAME="FixRedEye" VISIBLE="true"/>
                         <BUTTON NAME="Open" VISIBLE="true"/>
                         <BUTTON NAME="Grid" VISIBLE="true"/>
                         <BUTTON NAME="FlattenLayers" VISIBLE="true"/>
                         <BUTTON NAME="Help" VISIBLE="true"/>
                         <BUTTON NAME="ImageMenu" VISIBLE="true"/>
                         <BUTTON NAME="ImageRotateLeft" VISIBLE="true"/>
                         <BUTTON NAME="ImageRotateRight" VISIBLE="true"/>
                         <BUTTON NAME="ImageMirror" VISIBLE="true"/>
                         <BUTTON NAME="ImageCrop" VISIBLE="true"/>
                         <BUTTON NAME="ImageResample" VISIBLE="true"/>
                         <BUTTON NAME="ImageResizeCanvas" VISIBLE="true"/>
                         <BUTTON NAME="ColorMenu" VISIBLE="true"/>
                         <BUTTON NAME="ColorGrayScale" VISIBLE="true"/>
                         <BUTTON NAME="ColorBrightnessContrast" VISIBLE="true"/>
                         <BUTTON NAME="SharpenBlurMenu" VISIBLE="true"/>
                         <BUTTON NAME="InsertMenu" VISIBLE="true"/>
                         <BUTTON NAME="InsertImage" VISIBLE="true"/>
                         <BUTTON NAME="InsertRectangle" VISIBLE="true"/>
                         <BUTTON NAME="InsertEllipse" VISIBLE="true"/>
                         <BUTTON NAME="InsertLineArrow" VISIBLE="true"/>
                         <BUTTON NAME="InsertRichText" VISIBLE="true"/>
                         <BUTTON NAME="InsertTextAlongPath" VISIBLE="true"/>
                         <BUTTON NAME="HSL" VISIBLE="true"/>
                         <BUTTON NAME="EmbossLight" VISIBLE="true"/>
                         <BUTTON NAME="EmbossMedium" VISIBLE="true"/>
                         <BUTTON NAME="EmbossDark" VISIBLE="true"/>
                         <BUTTON NAME="OilPaint" VISIBLE="true"/>
                         <BUTTON NAME="WaterColor" VISIBLE="true"/>
                         <BUTTON NAME="Mosaic" VISIBLE="true"/>
                         <BUTTON NAME="Patchwork" VISIBLE="true"/>
                         <BUTTON NAME="BrickTexture" VISIBLE="true"/>
                       </BUTTONS>
                     </IMAGEEDITOR>
                     </PRESENTATIONOBJECT>
```

    e. Click **Save**.

3. Enable the Clarkii OIE instance as the attribute editor for an attribute of a given asset type. For this example, use the `Media` asset type's `FSII_ImageFile` attribute.

    a. Find and open the attribute asset in its Edit form. For this example, find and open the `FSII_ImageFile` attribute.

       • In the button bar, click **Search**.

- In the list of asset types, click the asset type of a attribute asset. For this example, click **Find Media Attribute**.

- In the **Search** field, enter the name of the attribute asset you want to modify. For this example, enter `FSII_ImageFile`.

- Click **Search**.

- In the list of search results, navigate to the attribute asset (`FSII_ImageFile`) and click its **Edit** (pencil) icon.

    WebCenter Sites opens the asset in its Edit form.

   b. Set Clarkii OIE as the attribute editor for this attribute asset:

- In the **Value Type** field, make sure **blob** is selected.

Note:

Clarkii OIE requires an attribute value of type `blob`. Once an attribute asset is saved, the value selected for the **Value Type** field cannot be modified.

- In the Attribute Editor drop-down list, choose the Clarkii OIE instance you created in step 2.

   c. Click **Save Changes**.

4. To test your new Clarkii OIE instance:

   a. Open the Contributor interface.

   b. Find any asset whose definition specifies the attribute you modified in step 3 as a field, and open that asset in its Edit form. For this example, select a `Media` asset:

- In the **Search** field, enter the search criteria.

- To narrow down your search to a specific asset type, click the down-arrow in the **Search** field to open the **Search Type** drop-down list. Click the asset type that is using the field enabled with the Clarkii OIE instance. For this example, select **Find Media**.

- Click the **magnifying glass** icon.

    A Search tab opens listing the results of your search.

- In the list of search results, right-click an asset of your choice. For this example, select the **FSII AudioCo_iAC-008.jpg** Media asset and then select **Edit**.

    WebCenter Sites displays the asset in its Edit form.

   c. Navigate to the field enabled with Clarkii OIE. For this example, **FSII_ImageFile** field. It should look similar to the following figure.

Figure 10-2   The Clarkii Online Image Editor
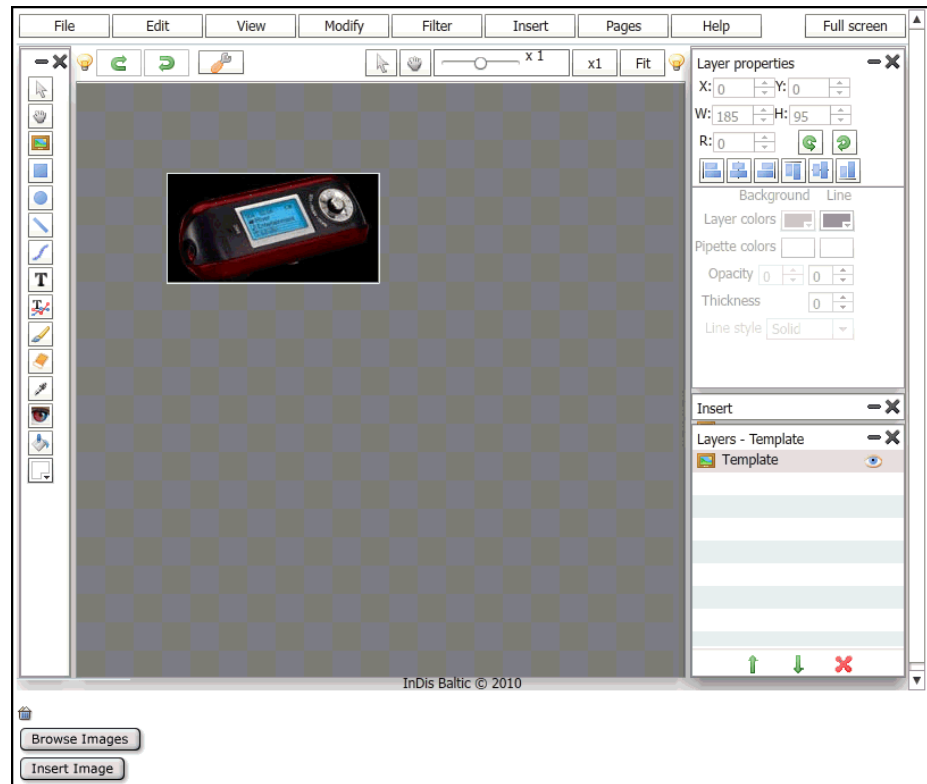
To render Clarkii OIE in the field, do one of the following:

- Make sure Flash is installed.

- Check the XML code of the Clarkii OIE instance you created (see, substep d of step 2).

- Check the selections you made in the attribute asset for which you enabled Clarkii OIE as the attribute editor (see substep b of step 3 of this procedure).

d. Click **Cancel** to return to the asset's Inspect form.

10.2.2 How to Configure Clarkii OIE Properties

The tables below list and define all of the properties that can be specified in the creation of a Clarkii OIE attribute editor asset. Use these tables as a reference to configure the properties for your own Clarkii OIE instance so it fits your site design.

Table 10-2   Clarkii OIE Specifications

Property	Definition
HEIGHT	Specify the height of the Clarkii OIE area as it will be displayed within the attribute field of a given asset's form. **Suggested value:** 600
WIDTH	Specify the width of the Clarkii OIE area as it will be displayed within the attribute field of a given asset's form. **Suggested value:** 800

Table 10-2 (Cont.) Clarkii OIE Specifications

Property	Definition
EDITORTYPE	Specify the type of image editor you want to use. To enable Clarkii OIE, the property must read: EDITORTYPE="clarkii"
FITIMAGE	If this property is set to true, images edited with Clarkii OIE will be resized to fit within the Clarkii OIE canvas. If this property is set to false, then when you edit an image with Clarkii OIE, that image is displayed on the canvas in its actual size. **Possible values:** true|false

Table 10-3 Browse Images Button Specifications

Property	Definition
ENABLEIMAGEPICKER	Enables the **Browse Images** button which allows users to place an image on the Clarkii OIE canvas, replacing any images that currently exist on the canvas. This button invokes the Image Picker window. **Possible values:** true|false **Note:** If this property is set to false, do not set values for the other properties related to the **Browse Images** button.
ASSETTYPE	Specify the asset type of the image assets that will be displayed in the Image Picker window when the **Browse Images** button is clicked. **Possible values:** Any asset type that has a definition containing an attribute of type blob which is intended to store images. **Example:** "Media_C"
ATTRIBUTE	Specify the image file attribute of the image assets that will be displayed in the Image Picker window when the **Browse Images** button is clicked. **Possible values:** Any attribute of type blob intended to store images. **Example:** "FSII_ImageFile"
ATTRIBUTETYPE	Specify the asset type of the image file attribute specified in the ATTRIBUTE property. Image Picker will look for attributes of only this asset type, and displays only the images with attributes of this asset type. **Possible values:** The asset type of the image file attribute specified in the ATTRIBUTE property. **Example:** "Media_A"

Table 10-3 (Cont.) Browse Images Button Specifications

Property	Definition
CATEGORYATTRIBUTE	Specify a `string` attribute holding a value that classifies the image assets (preferably by the names of their parent assets) to be displayed in the Image Picker window. The value of this property populates the **Category** drop-down list in the Image Picker with all values found for the specified attribute.
	When the **Browse Images** button is clicked, users can use the **Category** drop-down list to filter the images that are displayed in the Image Picker window by selecting one of these attribute values. Only the images matching the selected attribute value are rendered in the Image Picker window.
	Note: If no value is specified for this property, all image assets of the asset type specified in the ASSETTYPE property are displayed in the Image Picker window, and the **Category** drop-down list is not displayed.
RESTRICTEDCATEGORYLIST	In a comma-delimited list, enter specific values of the attribute specified in the CATEGORYATTRIBUTE property. The values you specify will be the only values available in the **Category** drop-down list of the Image Picker window when the **Browse Images** button is clicked.
	Possible values: Refer to the definition for the CATEGORYATTRIBUTE property.
	Note: If no value is specified for the CATEGORYATTRIBUTE property, then this property is ignored.

Table 10-4 Insert Image Button Specifications

Property	Definition
OIEATTRIBUTE	Specify the image file attribute of the image assets that will be displayed in the Image Picker window when the **Insert Image** button is clicked.
	Possible values: Any attribute of type `blob` intended to store images.
	Example: `"FSII_ImageFile"`
OIEATTRIBUTETYPE	Specify the asset type of the image file attribute you specified in the OIEATTRIBUTE property. Image Picker will look for attributes of only this asset type, and displays only the images with attributes of this asset type.
	Possible values: The asset type of the image file attribute specified in the OIEATTRIBUTE property.
	Example: `"Media_A"`
OIECATEGORYATTRIBUTE	Specify a `string` attribute holding a value that classifies the image assets (preferably by the names of their parent assets) that will be displayed in the Image Picker window. The value of this property populates the **Category** drop-down list in the Image Picker with all values found for the specified attribute.
	When the **Insert Image** button is clicked, users can use the **Category** drop-down list to filter the images that are displayed in the Image Picker window by selecting one of these attribute values. Only the images matching the selected attribute value are rendered in the Image Picker window.
	Note: If no value is specified for this property, all image assets of the asset type specified in the OIEASSETTYPE property are displayed in the Image Picker window, and the **Category** drop-down list is not displayed.

Table 10-4   (Cont.) Insert Image Button Specifications

Property	Definition
OIERESTRICTE DCATEGORYLIS T	In a comma-delimited list, enter specific values of the attribute specified in the OIECATEGORYATTRIBUTE property. The values you specify will be the only values available in the **Category** drop-down list of the Image Picker window when the **Insert Image** button is clicked.
	Possible values: Refer to the definition for the OIECATEGORYATTRIBUTE property.
	Note: If no value is specified for the OIECATEGORYATTRIBUTE property, then this property is ignored.
OIEENABLEIMA GEPICKER	Enables the **Insert Image** button, which enables users to insert an image as a layer on top of existing images on the Clarkii OIE canvas. This button invokes the Image Picker window.
	Possible values: true|false
	Note: If this property is set to false, do not set values for the other properties related to the **Insert Image** button.
OIEASSETTYPE	Specify the asset type of the image assets that will be displayed in the Image Picker window when the **Insert Image** button is clicked.
	Possible values: Any asset type that has a definition containing an attribute of type blob intended to store images.
	Example: "Media_C"

10.2.3 How to Implement a Field Copier Filter to Classify Assets

You use a field copier filter to classify image assets by the names of their parent assets. The following figure illustrates an example of how to implement a field copier filter, using the Media flex family of the FirstSiteII sample site. The field copier filter in this example copies the value of the system-defined Name attribute of the parent assets into a user defined string attribute.

To ensure that the values of preexisting attributes are not overwritten, create a new string attribute (FSII_ImageParentName) to hold the value of the system-defined attribute to be copied by the field copier. See Flex Filter Classes.

Figure 10-3   Sample Overview of a Field Copier Filter
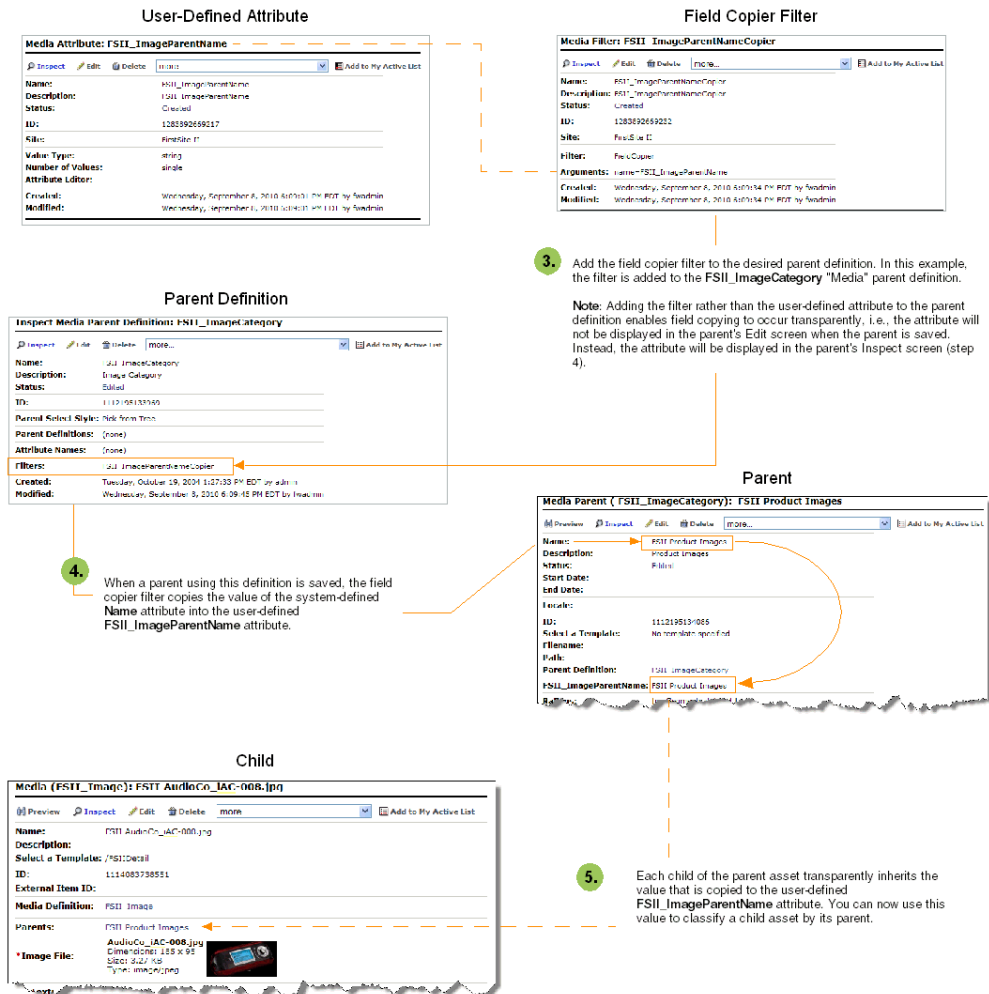
To Use a Field Copier Filter:

1.   Open the Admin interface.

2.   Determine the system-defined attribute whose value you want to use as the input for the field copier filter, and the user defined `string` attribute to which the field copier will be copying the system-defined attribute's value. To avoid overwriting the values of preexisting attributes, create a flex attribute of type `string` (`FSII_ImageParentName` in this example).

3.   Create a filter of type `FieldCopier` (`FSII_ImageParentNameCopier` in this example):

   a.   In the button bar, click **New**.

   b.   In the list of asset types, click **New Media Filter**.

   The New Media Filter form is displayed.

        c. In the **Name** field, enter a unique name for this filter (For example, `FSII_ImageParentNameCopier`).

        d. In the **Filter** field, choose **FieldCopier** from the drop-down list. Then click **Get Arguments**:

            • In the **Name** drop-down list, select the system-defined attribute whose value you want to use as the input value for the field copier filter (**name** in this example).

            • In the **Value** field, type the name of the user defined attribute (`FSII_ImageParentName` in this example) into which the field copier filter will copy the value of the system-defined attribute you specified.

        e. Click **Add** to add the argument to the filter.

        f. Click **Save** to save the filter.

4. Find a parent (or child) definition and add the new field copier filter to it (in this example we are adding the field copier filter to the `Media` parent definition `FSII_ImageCategory`):

        a. In the button bar, click **Search**.

        b. In the list of asset types, select **Find Media Parent Definition**.

        c. In the **Search** field, enter the name of the parent definition to which you want to add the field copier filter (`FSII_ImageCategory`).

        d. Click **Search**.

        e. In the list of search results, navigate to a parent (or child) definition and click its **Edit** (pencil) icon.

        f. In the Edit form of the parent (or child) definition, navigate to the **Filters** section and select the field copier filter you created in step 3.

        g. Click **Save Changes**.

5. Find and re-save all preexisting parent (or child) assets associated with the definition to which you added the field copier filter (`FSII_ImageParentNameCopier`). This enables the filter to populate the user defined attribute (`FSII_ImageParentName`) with the value of the system-defined attribute (`Name`).

If you added the filter to a parent definition, all children of the associated parent assets internally inherit the value that the field copier filter copied into the user defined attribute (`FSII_ImageParentName` in this example).

10.3 Configuring the Image Picker

When configuring an instance of the Image Picker attribute editor, you must specify the values of some parameters in the XML definition.

This table describes those parameters:

Table 10-5   Image Picker Parameters

Parameter	Explanation
ASSETTYPENAME	Asset type of the image assets that this instance of Image Picker will display. Example: `Media_C`
ATTRIBUTETYPENAME	Asset type of the image file attribute within the selected image asset type. Example: `Media_A`
ATTRIBUTENAME	Name of the image file attribute within the selected image asset type. Example: `FSII_ImageFile`
CATEGORYATTRIBUTENAME	(Optional) Name of the category attribute within the selected image asset type. Example: `FSII_ImageCategory`
RESTRICTEDCATEGORYLIST	Accepts a comma-delimited list of values of the category attribute within the selected image asset type. The values in this list will appear in the **Category** drop-down list in the Image Picker window. If this parameter is omitted, Image Picker will display all assets belonging to the selected image asset type. Example: `Audio,Video,Photo`

Sample XML code for an Image Picker definition is included in How to Create Image Picker Attribute Editor Definition Code. For instructions about creating new attribute editors (such as new instances of Image Picker) see Creating Attribute Editors. For instructions about selecting an input method (such as Image Picker) for a field in an asset form, see Create Flex Attributes in Creating a Flex Asset Family.

Topics:

- How to Categorize Image Assets for Display in Image Picker

- How to Create Image Picker Attribute Editor Definition Code

10.3.1 How to Categorize Image Assets for Display in Image Picker

Using the CATEGORYATTRIBUTENAME and RESTRICTEDCATEGORYLIST parameters described in the previous section, you can restrict an instance of Image Picker to display only selected categories of assets belonging to the selected image asset type. Before you do so, the following conditions must be satisfied:

- You must add a category attribute of type `string` to the selected image asset type or flex definition that will store the category descriptor for each asset within the selected asset type. Image Picker will use the value of this attribute to generate a list of asset categories which it is allowed display. For instructions about creating attributes, see Create Flex Attributes in Creating a Flex Asset Family.

- You or your content providers must fill in the category field for each asset of the selected image asset type, as appropriate. An asset which is not assigned a category is not displayed by a category-restricted instance of Image Picker.

10.3.2 How to Create Image Picker Attribute Editor Definition Code

A sample XML definition for the Image Picker attribute editor is included below for your reference. Use it to get an idea of how to configure Image Picker on your system.

```
<?XML VERSION="1.0"?>
<!DOCTYPE PRESENTATIONOBJECT>
<PRESENTATIONOBJECT NAME="ImagePicker">
<IMAGEPICKER>
  ASSETTYPENAME="Media_C"
  ATTRIBUTETYPENAME="Media_A"
  ATTRIBUTENAME="FSII_ImageFile"
  CATEGORYATTRIBUTENAME="FSII_ImageCategory"
  RESTRICTEDCATEGORYLIST="Audio,Video">
</IMAGEPICKER>
</PRESENTATIONOBJECT>
```

See Creating Attribute Editors.

11

Working with the WebCenter Sites Database

Just about everything in WebCenter Sites is represented as a row in a database table. The WebCenter Sites modules and products (Oracle WebCenter Sites: Engage, for example) deliver most of the tables you need. You work with the various kinds of tables and columns in the WebCenter Sites database.

For information about managing the data in non-asset tables, see Managing Data in Non-Asset Tables.

> **Note:**
>
> WebCenter Sites database tables used to be called *catalogs* and there are still remnants of that terminology throughout the application in table names, servlet names (CatalogManager), and the Java interfaces that you use to work with data in the database.

These topics can help you develop your own application or a table that does not hold assets (a lookup table, for example):

- Types of Database Tables

- Types of Columns (Fields)

- About Adding to the System Tables

- About Property Files and Databases

11.1 Types of Database Tables

In the WebCenter Sites database you use these tables: Object, Tree, Content, Foreign, and System.

- Object tables, which hold data as objects and provide a unique identifier, automatically, for each row in the table.

- Tree tables, which hold the hierarchical information about relationships between objects in object tables.

- Content tables, which hold flat data and do not provide a unique identifier for each row.

- Foreign tables, which can be either of the following:

  - Tables that are outside of the WebCenter Sites database but that WebCenter Sites has access to.

–   Tables that are in the WebCenter Sites database but that WebCenter Sites did not create.

- System tables, which are core WebCenter Sites application tables whose schema cannot be modified.

WebCenter Sites can cache the resultsets from queries against any table in the WebCenter Sites database, including foreign tables.

This section includes the following topics:

- Object Tables

- Tree Tables

- Content Tables

- Foreign Tables

- System Tables

- Identifying a Table's Type

11.1.1 Object Tables

Object tables store data as an object and can be represented in hierarchies. Those objects can be loaded, saved, and managed with the CatalogManager API. The asset type tables are object tables.

The primary key for object tables is always the ID (id) column and that cannot be changed. When you instruct WebCenter Sites to add an object table, it creates an ID column in that table. ID is a unique identifier assigned by default to each row as it is added to the table. For example, when someone creates a new asset, WebCenter Sites determines the ID and assigns that value as the ID for that asset.

You cannot change the ID assigned to objects (such as assets).

> **Note:**
>
> When AssetMaker or Flex Family maker creates an object table for a new asset type, it creates several additional columns by default. For information about the default columns in basic asset tables, see Default Columns in the Basic Asset Type Database Table.

WebCenter Sites handles ID generation for you, so use an object table to ensure that each row of the data you store is uniquely identified.

Examples of object tables (catalogs):

- All tables that hold assets

- Many of the publishing tables

- The Engage tables that hold visitor data

11.1.2 Tree Tables

Tree tables store information about the hierarchical relationships between object tables. In other words, object tables can be represented in hierarchies, but the hierarchy itself is stored in a tree table (the hierarchy is the tree).

For example, WebCenter Sites adds these tables to the WebCenter Sites database:

- `AssetRelationTree`: Stores information about associations between assets. These associations create parent-child relationships. For information about asset associations, see The Flex Asset Model.

- `SitePlanTree`: Stores information about parent-child relationships between page assets and the assets that are referred to from those assets. This information is presented graphically on the **Site Navigation** tab that is present in the WebCenter Sites interface.

Each row in a tree table is a node in that tree. Each node in a tree table points to two places:

- To an object in an object table, that is, to the object that it represents.

- To its parent node in that tree table, unless it is a top-level node and has no parent.

In other words, the object itself is stored in an object table. Its relationships to other objects in the database (as described by the tree) are stored in the tree table as a node on a tree. Children nodes point to parent nodes but parents do not point to children.

When you create a tree table, it has columns described in Table 11-1 by default. You cannot modify these columns or add new ones.

Table 11-1   Default Columns

Column	Description
nid	The ID of the node. This is the primary key.
nparentid	The ID of the node's parent node.
nrank	A number that ranks peer or sibling nodes. For example, the `AssetRelationTree` table uses this column to determine the order of the assets that are in collections.
otype	The object type of the node. For example, in the `SitePlanTree` table, `otype` is either the asset type `page` or the name of a site (`publication`). In the `AssetRelationTree` table, `otype` is an asset type and is the name of the object table for assets of that type.
oid	The ID of the object that the node refers to.
oversion	Reserved for future use.
ncode	Holds a string that has meaning in the context of what the table is being used for. For example, in the `SitePlanTree`, `ncode` is set to `placed` or `unplaced` based on whether the page asset that the node refers to has been placed or not. In the `AssetRelationTree`, `ncode` holds the name of an association.

11.1.3 Content Tables

Content tables store data as flat data (rather than as objects) and that information cannot be organized in a hierarchy. You use content tables for simple lookup tables. For example, these are only a few of the content tables that add to the WebCenter Sites database:

- `Source`: Holds strings that are used to identify the source of an article or image asset.

- `Category`: Holds codes that are used to organize assets in several ways.

- `StatusCode`: Holds the codes that represent the status of an asset.

All three of these tables are lookup tables that the product uses to look up values for various columns in the asset type tables (object tables).

In another example, WebCenter Sites also adds a content table called `MimeType`. This table holds mimetype codes that are displayed in the **Mimetype** fields of the stylesheet and imagefile asset types. The **Mimetype** fields for these asset types query the `MimeType` table for mimetype codes based on the `keyword` column in that table.

Setting the Primary Key for a Content Table

When you create a content table, an ID column is not created for you and the primary key is not required to be ID. This is another major difference between content tables and object tables.

The `cc.contentkey` property in the `wcs_properties.json` file specifies the name of the default primary key for all content tables. When you create a new content table, you are responsible for defining a column with the name specified by the `cc.contentkey` property. However, you can override the identity of the primary key for a specific content table by adding and setting a custom property in the `wcs_properties.json` file. This property must use the following format:

`cc.tablenameKey`

For example, to create a content table named `Books` and to override the default primary key so that it uses the `ISBN` column instead, add a property named `cc.BooksKey` and set it to `ISBN`.

11.1.4 Foreign Tables

A foreign table is one that WebCenter Sites does not completely manage. For example, a site's pages perform queries against a table that is populated by an ERP system and WebCenter Sites displays that information to the site visitors.

WebCenter Sites can query foreign tables and cache the resultsets just as it does for its own object and content tables. However, you must first identify that foreign table to WebCenter Sites by adding a row for it in the `SystemInfo` table. This is the only time you should ever modify information in the `SystemInfo` table. Additionally, you must be sure to flush the WebCenter Sites resultset cache with a CatalogManager `flushcatalog` tag whenever the external system updates the tables that you query. Otherwise, the resultsets cached against those tables might not be up-to-date.

For information about resultset caching, see Working with Resultset Caching and Queries.

11.1.5 System Tables

System tables are core, WebCenter Sites tables whose schema is fixed. They are implemented in WebCenter Sites by their own classes and they do not follow the rules (for caching and so on) that the other tables follow.

You can add rows to some system tables using the Explorer tool, but you cannot add or modify the columns in these tables in any way. You also cannot add system tables to the database.

This table lists and defines the WebCenter Sites system tables:

Table 11-2   System Tables

Table	Description
ElementCatalog	Lists all the XML or JSP elements used in your system. An element is a named piece of code.
SiteCatalog	Lists a page reference for each page or pagelet served by WebCenter Sites.
SystemACL	Has a row for each of the access control lists (ACLs) that were created for your WebCenter Sites system. ACLs are sets of permissions to database tables.
SystemEvents	Has a row for each event being managed by WebCenter Sites. An event represents an action that takes place on a certain schedule. WebCenter Sites inserts a row in this table when you set an event by using either the APPEVENT or EMAILEVENT tags.
SystemInfo	Lists all the tables that are in the WebCenter Sites database and any foreign tables that WebCenter Sites needs to reference.
SystemSeedAccess	Registers Java classes that are external to WebCenter Sites but that WebCenter Sites has access to (includes access control).
SystemSQL	Holds SQL queries that you can reuse in as many pages or pagelets as necessary. You can store SQL queries in this table and then use the ics.CallSQL method, CALLSQL XML tag, the ics:callsql JSP tag to invoke them. Then, you have to modify the SQL statement only once, if required.
SystemUserAttr	Stores attribute information about the users such as their email addresses. Note that this table is not used for LDAP.
SystemUsers	Lists all the users who are allowed access to pages, functions, and tables. Note that this table is not used for LDAP.

11.1.6 Identifying a Table's Type

To determine the table type of any table in the WebCenter Sites database, examine the SystemInfo table, the system table that lists all the tables in the database.

To Determine a Table Type:

1. Open Explorer and log in to the WebCenter Sites database.

2. Double-click the `SystemInfo` table.

3. In the list of tables, examine the `systable` column. The value in this column identifies the type of table represented in Table 11-3.

Table 11-3   Determining Table Types

Value in systable column	Definition
`yes`	system table
`no`	content table
`obj`	object table
`tree`	tree table
`fgn`	foreign table

Note:

You cannot open and examine the `SystemInfo` table without the appropriate ACLs assigned to your user name.

11.2 Types of Columns (Fields)

In the tables you create for the WebCenter Sites database, you can specify these field (column) types for the columns: Generic, Database-specific, and WebCenter Sites URL.

See these topics:

- Generic Field Types
- Database-Specific Field Types
- Indirect Data Storage with the WebCenter Sites URL Field

11.2.1 Generic Field Types

Generic field types refer to field types that work in any DBMS that WebCenter Sites supports. They are mapped to be compliant with JDBC standards. Therefore, if your WebCenter Sites system changes to a different DBMS, your database is still valid.

With generic, JDBC-compliant field types, use the CatalogManager API (`CATALOGMANAGER` XML or JSP tags, or the `ics.CatalogManager` Java method) to modify and maintain the data in your tables.

The following table contains a complete list of the WebCenter Sites generic field types and the database properties (from the `wcs_properties.json` file) that define their data types. Refer to this list whenever you create a new table with the Explorer tool or the CatalogManager API.

Table 11-4   Field Types

Field Type	Description	Property
`CHAR(n)`	A short string of exactly n characters.	`cc.char`

Table 11-4   (Cont.) Field Types

Field Type	Description	Property
VARCHAR(n)	A short string of up to n characters. For example, VARCHAR(32) means that this column can hold a string of up to 32 characters.	cc.varchar and cc.maxvarcharsize (The maximum value that you can set for cc.varchar depends on the value of the cc.maxvarcharsize property.)
DATETIME	A date/time combination.	cc.datetime
TEXT	A LONGVARCHAR, a variable-length string of up to 2,147,483,647.	cc.bigtext
IMAGE	One binary large object (blob).	cc.blob
SMALLINT	A 16-bit integer, that is, an integer from -32,768 to +32,767.	cc.smallint
INTEGER	A 32-bit integer, that is, an integer from -2,147,483,648 to +2,147,483,647.	cc.integer
BIGINT	A 64-bit integer, that is, integers having up to 19 digits.	cc.bigint
NUMERIC(L,P)	A floating-point (real) number, having a total number of L significant digits of which up to P significant digits are fractional. For example, NUMERIC(5,2) could represent a number such as 806.35 but could not accurately represent a number such as 25693.2283.	cc.numeric
DOUBLE	A double precision type.	cc.double

In addition to defining the column type, you must specify which of the column constraints, as described in Table 11-5, applies to the column.

Table 11-5   Column Constraints

Constraint	Description
NULL	It can hold a null value, that is, it can be left empty.
NOT NULL	It cannot hold a null value, that is, it cannot be left empty.
UNIQUE NOT NULL	It must hold a value that is guaranteed to be unique in this table.
PRIMARY KEY NOT NULL	Marks the primary key column in a content table. You cannot set this column constraint for an object table.

When you use AssetMaker to create an object table for a new asset type or when you create flex attributes, the data types for those items are different than the ones listed here.

See Storage Types for the Columns and Data Types for Attributes.

11.2.2 Database-Specific Field Types

You can use database-specific field (column) types in your tables. However, if you use field types that are specific to one kind of DBMS (that is, types that have not been mapped to a JDBC standard), note the following:

- You may not be able to use the CatalogManager API on those tables.

- If you ever change your DBMS you must also modify your tables.

For a complete list of field types specific to the DBMS that you are using, consult your DBMS documentation.

11.2.3 Indirect Data Storage with the WebCenter Sites URL Field

Object and content tables in the WebCenter Sites database have a unique characteristic that columns can store their data indirectly. You can store large bits of data externally to the DBMS but within the data repository.

To create such a column, you must use a column name that begins with the letters url. When you use the letters url as the first three letters of a column name, WebCenter Sites treats that column as an indirect data column.

Use a URL field for the following reasons:

- When the DBMS you are using does not support fields that are large enough to accommodate the size of the data that you want to store there.

- If the DBMS you are using does not support enough fields in an individual table to contain the data that you want to store.

- Because the performance of selecting data degrades with large field sizes.

> **Note:**
>
> If the size of the data you want to store in a URL column exceeds the value set for the cc.maxvarcharsize property in the wcs_properties.json file, then that data is stored in the database, instead of being stored indirectly as a file that is referenced by a pointer in the database.

The Default Storage Directory (defdir)

Any table with a URL column must have a default storage directory specified for it. This directory is where the values entered into the column are actually stored.

The phrase *default storage directory* is shortened to the word *defdir* in several places in the product. For example, the defdir column in the SystemInfo table holds the name of the default storage directory for tables with URL columns; one of the forms for the AssetMaker utility presents a defdir field, and so on.

The value entered into a URL field is actually a relative path to a file because this value is appended to the value of the table's defdir setting. The way that you set the

`defdir` value for the tables that you create depends on the applications you have and what you are doing:

- To create a new WebCenter Sites table with the CatalogManager API, use the `uploadDir` argument to set the value of `defdir`.

- To create a new basic asset type, specify the value of the `defdir` in the **defdir** field on the AssetMaker form. Note that all tables that hold basic assets have a URL column and must have a `defdir` value set.

- To create a new flex asset type, do not specify the value of the `defdir` for the URL column in the flex asset's _Mungo table. This value is obtained from a property that was set when your WebCenter Sites application was installed. Never change the value of that property.

> **Note:**
>
> After a table with a URL column is created, do not attempt to change or modify the `defdir` setting for the table in any way. If you do, the link between the storage directory and the URL column is broken, and your data can no longer be retrieved.

For information about creating URL fields, see the following procedures and examples:

- The upload field examples for basic asset types, starting with Example 5-3 in Creating a Flex Asset Family.

- The upload field example for creating flex attributes of type `blob` (Creating Flex Attributes of Type Blob (Upload Field)) in Creating a Flex Asset Family.

11.3 About Adding to the System Tables

You cannot create system tables, but you can add rows to some of them with the Explorer tool. How you add information to each of these tables is different.

Table 11-6   Methods of Adding Information to System Tables

Table	Method of Adding Information
SiteCatalog	There are several ways that page entries are added to this table: • When you create a `Template` asset, WebCenter Sites automatically creates a page entry for it in the `SiteCatalog` table. • When you create a `SiteEntry` asset, WebCenter Sites automatically creates a page entry for it in the `SiteCatalog` table. To set or modify page cache settings for page entries, it is easier to use forms in the WebCenter Sites interface than it is to use Explorer.
ElementCatalog	There are several ways that elements are added to this table: • When you create a `Template` asset, WebCenter Sites automatically creates an entry for it in the `ElementCatalog` table. • When you create a `CSElement` asset, WebCenter Sites automatically creates an entry for it in the `ElementCatalog` table. • You can use the Explorer tool to add non-asset elements. For information about coding elements and pages, see Coding Elements for Templates and CSElements.

Table 11-6   (Cont.) Methods of Adding Information to System Tables

Table	Method of Adding Information
SystemACL	The ACL form in the **User Access Management** node.
SystemEvents	WebCenter Sites adds a row to this table for each event that is designated when an APPEVENT tag, EMAILEVENT tag, or Java API equivalent is invoked from an element.
SystemInfo	Do not add or modify information to this table. The only exception to this rule is if you have to identify a foreign table to WebCenter Sites.
SystemSQL	The Explorer tool. For information about the various kinds of queries that are available, see Working with Resultset Caching and Queries.
SystemUsers	The User form in the **User Access Management** node.
SystemUserAttr	The User form in the **User Access Management** node.

11.4 About Property Files and Databases

Database properties in the wcs_properties.json file help you configure the WebCenter Sites database connection. You use these properties to establish a privileged and non-privileged user connection between the database and the application server.

The database properties were configured for your system when you installed WebCenter Sites. By default, all commands identified in the wcs_properties.json file operate on the WebCenter Sites database. To access the properties in the wcs_properties.json file, use the Property Management Tool in the WebCenter Sites Admin interface.

12

Managing Data in Non-Asset Tables

How do you interact with WebCenter Sites database tables that do not hold assets? You can work with the data in your custom, non-asset tables programmatically and manually. You use tags and methods for the CatalogManager API to code forms for data entry and management. And, through the Explorer tool you manually add rows and data to those rows.

To work with assets, you must log in to the WebCenter Sites interface and use the asset forms provided by the WebCenter Sites and Oracle WebCenter Sites: Engage applications. To add large numbers of assets programmatically, use the XMLPost utility, as described in Importing Assets of Any Type and Importing Flex Assets.

Topics:

- Using Methods and Tags to Program Data Management in Non-Asset Tables
- Coding Data Entry Forms
- Consideration About Deleting Non-Asset Tables

12.1 Using Methods and Tags to Program Data Management in Non-Asset Tables

Would you like to program how you manage and interact with non-asset tables? Java methods such as CatalogManager and TreeManager, and XML tags such as CATALOGMANAGER and TREEMANAGER are available to help you do just that.

See these topics:

- About Writing and Retrieving Data
- Methods for Querying for Data
- Lists and Listing Data

12.1.1 About Writing and Retrieving Data

CatalogManager is the WebCenter Sites servlet that manages content and object tables in the database. The TreeManager servlet manages tree tables in the database.

- To access the CatalogManager servlet, use the `ics.CatalogManager` Java method, the `CATALOGMANAGER` XML tag, or the `ics:catalogmanager` JSP tag.
- To access the TreeManager servlet, use the `ics.TreeManager` Java method, the `TREEMANAGER` XML tag, or the `ics:treemanager` JSP tag.

These methods and tags take name/value pairs from arguments that specify the operation to perform and the table on which to perform that operation.

12.1.1.1 Security Through CatalogManager

The `ics.CatalogManager` Java method, the `CATALOGMANAGER` XML tag, and the `ics:catalogmanager` JSP tag support several attributes that operate on object and content tables. The key attribute is `ftcmd`. By setting `ftcmd` to `addrow`, for example, you tell CatalogManager to add one row to the catalog.

CatalogManager security, when enabled, prevents users with the `DefaultReader` ACL from accessing CatalogManager. You enable CatalogManager security by setting the `secure.CatalogManager` property, found in the `wcs_properties.json` file, to `true`. Note that your session will be dropped if you attempt to log out of CatalogManager when CatalogManager security is enabled.

The table below shows the main `CATALOGMANAGER` XML tag attributes. They are passed as argument name/value pairs that modify the contents of a row or a particular field in a row.

Table 12-1   CATALOGMANAGER XML Tag

argument name="ftcmd" value=	Description
addrow	Adds a single row to a table.
addrows	Adds multiple rows to a table.
deleterow	Deletes a row from a table. You must specify the primary key column for the row.
deleterows	Deletes multiple rows from a table. You must specify the primary key for the rows.
replacerow	Deletes the existing row in a table and replaces the row with the specified information.
replacerows	Replaces multiple rows in a table. If a value is not specified for a column, the column value is cleared.
updaterow	Performs a query against a given table and displays records from a table. The rows displayed match the criteria specified by the value of the parameters.
updaterow2	Like `updaterow`, updates values in columns for a row in a table. However, where you cannot clear columns with `updaterow`, `updaterow2` lets you clear columns if there is no value for the specified column (for example, if there is no related field in the form).
updaterows	Modifies field values for multiple rows in a table.
updaterows2	Like `updaterows`, modifies field values for multiple rows in a table. However, where you cannot clear columns with `updaterows`, `updaterows2` lets you clear them if there is no value for the specified column (for example, if there is no related field in the form).

Any requests going to CatalogManager with the command parameter (`ftcmd`) must be either a `POST` request or one of the following commands:

- exportlog

- exportForm

- logout

- selectFromTable

- selectCount

- mirrorgetconfig

- listtables

- retrieve

- retrievebinary

- pingdb

- interrogatetbl

- checksession

- history

- retrieverevision

For more information and a complete list of the CatalogManager commands, see the *Oracle Fusion Middleware Tag Reference for Oracle WebCenter Sites Reference*. For information about the `ics.CatalogManager` Java method, see the *Oracle Fusion Middleware Java API Reference for Oracle WebCenter Sites*.

12.1.1.2 Tree Manager Commands for Managing the Tree Tables

The table below shows the main `ics.TreeManager` commands. Note that these operations manipulate data in the tree table only, but do not affect the objects that the tree table nodes refer to.

Table 12-2   TreeManager Commands

Name	Description
addchild	Given a parent node, adds a child node.
addchildren	Adds multiple child nodes.
copychild	Copies a node and its children to a different parent. All copied nodes point to the same objects.
createtree	Creates a tree table.
delchild	Deletes a node and its child nodes.
delchildren	Deletes multiple nodes.
deletetree	Deletes a tree table.
findnode	Finds a node in a tree.
getchildren	Gets all child nodes.

Table 12-2   (Cont.) TreeManager Commands

Name	Description
getnode	Gets node and optionally object attributes.
getparent	Gets the nodes parent.
listtrees	Gets the list of all tree tables.
movechild	Moves node and its child nodes to a different parent.
nodepath	Returns parent; child path to a node.
setobject	Associates a different object with the node.
validatenode	Verifies that a node is in a tree.
verifypath	Verifies that a given path exists in a tree.

For information about the ics.TreeManager method, see the *Oracle Fusion Middleware Java API Reference for Oracle WebCenter Sites*.

For information about the XML and JSP TREEMANAGER tags, see the *Oracle Fusion Middleware Tag Reference for Oracle WebCenter Sites Reference*.

12.1.2 Methods for Querying for Data

This table shows the three methods, with XML and JSP tag counterparts, to help your code query for and select content:

Table 12-3   Querying for Data

Method	XML tag	JSP tag	Description
ics.SelectTo	SELECTTO	ics:selectto	Performs a simple select against a single table.
ics.SQL	EXECSQL	ics:sql	Executes an inline SQL statement (embedded in the code).
ics.CallSQL	CALLSQL	ics:callsql	Executes a SQL statement that is stored as a row in the SystemSQL table.

To use ics.CallSQL (or the tags), you code SQL statements and then paste them into the SystemSQL table. By storing the actual queries in the SystemSQL table and calling them from the individual pages (like you call a pagename or an element), you keep them out of your code, which makes it easier to maintain the SQL used by your site. To change the SQL, you do not have to fix it in every place that you use it, just edit it in the SystemSQL table so every element calls the edited version.

The ics.CallSQL and ics.SQL methods can execute any legal SQL commands. If a SQL statement does not return a usable list, WebCenter Sites will generate an error. To update or insert data using SQL, you must include code that explicitly flushes the resultsets cached against the appropriate tables using the ics.FlushCatalog method.

12.1.3 Lists and Listing Data

Several `ICS` methods create lists. The `SelectTo` method, for example, returns the results of a simple SQL query in a list whose columns reflect the items in the `WHAT` clause and whose rows reflect matches against the table.

The `IList` interface is used to access a list from Java. The lists are available by name using XML or JSP, and values can be iterated using the `LOOP` tag. The lists created by WebCenter Sites point to underlying resultsets created from querying the database. Although the lists do not persist across requests, the resultsets do if they are cached.

> **Note:**
>
> Be sure to configure resultset caching appropriately. The list points to a copy of the query's cached resultset. If the resultset is not cached, the list points directly at the resultset which can cause database connection resource difficulties.

You can create your own list for use in XML or JSP by implementing a class based on the IList interface. Your application or page can transform data before returning an item in a list or to create a single list from many lists. This table shows the methods that manage lists:

Table 12-4   Methods that Manage Lists

Method	Description
ics.GetList	Returns an `IList`, given the name of the list.
ics.CopyList	Copies a list.
ics.RenameList	Renames an existing list.
ics.RegisterList	Registers a list by name with WebCenter Sites so that you can reference the list from an XML or JSP element or by using the `GetList` method.

For an example implementation of an `IList`, see `SampleIList.java` in the `Samples` folder on your WebCenter Sites system.

12.2 Coding Data Entry Forms

We have code samples to show you how you can code forms in which site visitors enter information, and how that information should be stored in the database. Use these code samples to learn how to add a new row, run a query for a row, and edit or delete a row. Each sample shows a version for XML, JSP, and Java.

See these topics:

- How To Add a Row

- How To Delete a Row

- How To Query a Table

- How To Query a Table with an Embedded SQL Statement

12.2.1 How To Add a Row

A simple algorithm for adding a row is as follows:

1. Display a form requesting information for each of the fields in a row.

2. Write that form data to the table.

This example adds a row to a fictitious table named `EmployeeInfo` with the columns shown in the following table:

Table 12-5   Example Adds Row to Table

Field	Data type
id	VARCHAR(6)
phone	VARCHAR(16)
name	VARCHAR(32)

This example presents code from the following elements:

- `addrowFORM`, an XML element that displays a form that requests an employee ID number, phone number, and name.

- `addrowXML`, `addrowJSP`, and `addrowJAVA`, three versions of an element that writes the information entered by the employee to the `EmployeeInfo` table.

12.2.1.1 The addrowFORM Element

The `addrowFORM` element displays a form that asks the user to enter employee name, ID, and phone.

This is the code that creates the form:

```
<?xml version="1.0" ?>
<!DOCTYPE FTCS SYSTEM "futuretense_cs.dtd">
<FTCS Version="1.1">
<!-- Documentation/CatalogManager/addrowFORM
-->

<form ACTION="ContentServer" method="post"
REPLACEALL="CS.Property.ft.cgipath">
<input type="hidden" name="pagename" value="Documentation/CatalogManager/addrow"/>

<table>
<tr>
<td>Employee name:</td>
<td><input type="text" value="" name="EmployeeName" size="22" maxlength="32"/></td>
</tr>
<tr>
<td>Employee id number:</td>
<td><input type="text" value="" name="EmployeeID" size="6" maxlength="6"/></td>
</tr>
<tr>
<td>Phone number:</td>
<td><input type="text" value="" name="EmployeePhone" size="12" maxlength="16"/></td>
</tr>
```

```
<tr>
<td colspan="2"><input type="submit" name="submit" value="Submit"/></td>
</tr>
</table>

</form>
</FTCS>
```

Notice that the `maxlength` modifiers in `<INPUT>` limit the length of each input to the maximum length that was defined in the schema.

The user fills in the form and clicks the **Submit** button. The information gathered in the form and the pagename of the `addrow` page (see the first `input type` statement in the preceding code sample) is sent to the browser. The browser sends the pagename to WebCenter Sites. WebCenter Sites looks it up in the `SiteCatalog` table and then invokes that page entry's root element.

12.2.1.2 Root Element for the addrow Page

The root element of the `addrow` page is responsible for adding the information passed from the `addrowFORM` element to the database. That is, for adding a row to the `EmployeeInfo` table and populating that row with the information passed from the `addrowFORM` element.

There can only be one root element for a WebCenter Sites page (that is, an entry in the `SiteCatalog` table). This section shows three versions of the root element for the `addrow` page:

- `addrowXML.xml`

- `addrowJSP.jsp`

- `addrowJAVA.jsp`

addrowXML

This is the code in the XML version of the root element:

```
<?xml version="1.0" ?>
<!DOCTYPE FTCS SYSTEM "futuretense_cs.dtd">
<FTCS Version="1.1">
<!-- Documentation/CatalogManager/addrowXML
-->

<SETVAR NAME="errno" VALUE="0"/>
<CATALOGMANAGER>

<ARGUMENT NAME="ftcmd" VALUE="addrow"/>
<ARGUMENT NAME="tablename" VALUE="EmployeeInfo"/>
<ARGUMENT NAME="id" VALUE="Variables.EmployeeID"/>
<ARGUMENT NAME="phone" VALUE="Variables.EmployeePhone"/>
<ARGUMENT NAME="name" VALUE="Variables.EmployeeName"/>

</CATALOGMANAGER>
errno=<CSVAR NAME="Variables.errno"/><br/>
</FTCS>
```

> **Note:**
>
> The example code can use the CATALOGMANAGER tag because the fictitious table, EmployeeInfo, has WebCenter Sites generic field types. addrowXML might not work if EmployeeInfo has database-specific field types. See Generic Field Types.

addrowJSP

This is the code in the JSP version of the root element:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld" %>
<%@ taglib prefix="ics" uri="futuretense_cs/ics.tld" %>
<%//
// Documentation/CatalogManager/addrowJSP
//%>
<%@ page import="COM.FutureTense.Interfaces.*" %>
<%@ page import="COM.FutureTense.Util.ftMessage"%>
<%@ page import="COM.FutureTense.Util.ftErrors" %>
<cs:ftcs>

<ics:setvar name="errno" value="0"/>
<ics:catalogmanager>

<ics:argument name="ftcmd" value="addrow"/>
<ics:argument name="tablename" value="EmployeeInfo"/>
<ics:argument name="id"
value='<%=ics.GetVar("EmployeeID")%>'/>
<ics:argument name="phone"
value='<%=ics.GetVar("EmployeePhone")%>'/>
<ics:argument name="name" value='<%=ics.GetVar("EmployeeName")%>'/>

</ics:catalogmanager>

errno=<ics:getvar name="errno"/><br/>

</cs:ftcs>
```

addrowJAVA

This is the code in the Java version of the root element:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld" %>
<%//
// Documentation/CatalogManager/addrowJAVA
//%>
<%@ page import="COM.FutureTense.Interfaces.*" %>
<%@ page import="COM.FutureTense.Util.ftMessage"%>
<%@ page import="COM.FutureTense.Util.ftErrors" %>
<cs:ftcs>

<!-- user code here -->
<%
ics.SetVar("errno","0");
FTValList vl = new FTValList();
vl.put("ftcmd","addrow");
vl.put("tablename","EmployeeInfo");
vl.put("id",ics.GetVar("EmployeeID"));
vl.put("phone",ics.GetVar("EmployeePhone"));
vl.put("name",ics.GetVar("EmployeeName"));
```

```
ics.CatalogManager(vl);
%>
errno=<%=ics.GetVar("errno")%><br />

</cs:ftcs>
```

12.2.2 How To Delete a Row

The following example deletes a row from the fictitious `EmployeeInfo` table.

This section presents code from the following elements:

- `deleterowFORM`: An XML element that displays a form that requests an employee name to delete from the `EmployeeInfo` table.

- `deleterowXML`, `deleterowJSP`, `deleterowJAVA`: Elements that delete a row from the `EmployeeInfo` table based on the information sent to it from the `deleterowFORM` element.

12.2.2.1 The deleterowFORM Element

The `deleterowFORM` element displays a form that asks the user to enter an employee name. This is the code that creates the form:

```
<?xml version="1.0" ?>
<!DOCTYPE FTCS SYSTEM "futuretense_cs.dtd">
<FTCS Version="1.1">
<!-- Documentation/CatalogManager/deleterowFORM
-->

<form ACTION="ContentServer" method="post" REPLACEALL="CS.Property.ft.cgipath">
<input type="hidden" name="pagename" value="Documentation/CatalogManager/deleterow"/>

<table>
<tr>

<td>Employee name:</td>
<td><input type="text" value="Barton Fooman" name="EmployeeName" size="22"
maxlength="32"/></td>

</tr>
<tr>

<td colspan="2"><input type="submit" name="submit" value="submit"/></td>

</tr>
</table>
</form>
</FTCS>
```

The user enters an employee name and clicks the **Submit** button. The employee name and the pagename for the `deleterow` page (see the first `input type` statement in the preceding code sample) are sent to the browser. The browser sends the pagename to WebCenter Sites. WebCenter Sites looks it up in the `SiteCatalog` table and then invokes that page entry's root element.

12.2.2.2 Root Element for the deleterow Page

The root element of the `deleterow` page is responsible for deleting a row from the `EmployeeInfo` table, based on the employee name that is sent to it from the `deleterowFORM` element. There can only be one root element for a WebCenter Sites

page (that is, an entry in the `SiteCatalog` table). This section shows three versions of the root element for the `deleterow` page:

- `deleterowXML.xml`

- `deleterowJSP.jsp`

- `deleterowJAVA.jsp`

deleterowXML

This is the code in the XML version of the element:

```
<?xml version="1.0" ?>
<!DOCTYPE FTCS SYSTEM "futuretense_cs.dtd">
<FTCS Version="1.1">
<!-- Documentation/CatalogManager/deleterowXML
-->
<SETVAR NAME="errno" VALUE="0"/>

<CATALOGMANAGER>

<ARGUMENT NAME="ftcmd" VALUE="deleterow"/>
<ARGUMENT NAME="tablename" VALUE="EmployeeInfo"/>
<ARGUMENT NAME="tablekey" VALUE="name"/>
<ARGUMENT NAME="tablekeyvalue" VALUE="Variables.EmployeeName"/>

</CATALOGMANAGER>

errno=<CSVAR NAME="Variables.errno"/><br/>
</FTCS>
```

deleterowJSP

This is the code in the JSP version of the element:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld" %>
<%@ taglib prefix="ics" uri="futuretense_cs/ics.tld" %>
<%//
// Documentation/CatalogManager/deleterowJSP
//%>
<%@ page import="COM.FutureTense.Interfaces.*" %>
<%@ page import="COM.FutureTense.Util.ftMessage"%>
<%@ page import="COM.FutureTense.Util.ftErrors" %>
<cs:ftcs>

<!-- user code here -->
<!-- user code here -->
<ics:setvar name="errno" value="0"/>
<ics:catalogmanager>

<ics:argument name="ftcmd" value="deleterow"/>
<ics:argument name="tablename" value="EmployeeInfo"/>
<ics:argument name="name" value='<%=ics.GetVar("EmployeeName")%>'/>

</ics:catalogmanager>

errno=<ics:getvar name="errno"/><br />

</cs:ftcs>
```

deleterowJAVA

This is the code in the Java version of the element:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld" %>
<%//
// Documentation/CatalogManager/deleterowJAVA
//%>
<%@ page import="COM.FutureTense.Interfaces.*" %>
<%@ page import="COM.FutureTense.Util.ftMessage"%>
<%@ page import="COM.FutureTense.Util.ftErrors" %>
<cs:ftcs>

<%
ics.SetVar("errno","0");
FTValList vl = new FTValList();
vl.put("ftcmd","deleterow");
vl.put("tablename","EmployeeInfo");
vl.put("name",ics.GetVar("EmployeeName"));
ics.CatalogManager(vl);
%>
errno=<%=ics.GetVar("errno")%><br />

</cs:ftcs>
```

12.2.3 How To Query a Table

The following sample elements query the fictitious `EmployeeInfo` table for an employee's name, extract the employee name and displays it in a browser, prompts the user to edit the information, and then writes the edited information to the database.

This section presents code from the following elements:

- `SelectNameForm`, an XML element that displays a form that requests an employee's name.

- Three versions of the `QueryEditRowForm` element (XML, JSP, and Java), an element that locates the employee name and loads the information about that employee into a form that the employee can use to edit his or her information.

- Three versions of the `QueryEditRow` element (XML, JSP, and Java), an element that writes the newly edited information to the database.

12.2.3.1 The SelectNameForm Element

The `SelectNameForm` element displays a simple form that requests the name of the employee who is altering employee information. This is the code:

```
<?xml version="1.0" ?>
<!DOCTYPE FTCS SYSTEM "futuretense_cs.dtd">
<FTCS Version="1.1">
<!-- Documentation/CatalogManager/SelectNameForm
-->

<form ACTION="ContentServer" method="post">
<input type="hidden" name="pagename" value="Documentation/CatalogManager/
QueryEditRowForm"/>
<TABLE>
<TR>
<TD>Employee name: </TD>
```

```
<TD><INPUT type="text" value="" name="EmployeeName" size="22" maxlength="32"/></TD>
</TR>
<TR>
<TD COLSPAN="100%" ALIGN="CENTER">
<input type="submit" name="doit" value="Submit"/></TD>
</TR>
</TABLE>
</form>
</FTCS>
```

When the employee clicks the **Submit** button, the information gathered in the **Employee name** field and the name of the `QueryEditRowForm` page (see the first `input type` statement in the preceding code sample) is sent to the browser. The browser sends the page name to WebCenter Sites. WebCenter Sites looks up the page name in the `SiteCatalog` table, and then invokes that page entry's root element, `QueryEditRowForm`.

12.2.3.2 The Root Element for the QueryEditRowForm Page

The root element for the `QueryEditRowForm` page locates the row in the `EmployeeInfo` table that matches the string entered in the **Employee name** field and then loads the data from that row into a new form. The employee can edit the name and phone number but cannot edit the ID number.

The Change Employee Information form looks like this:

Figure 12-1   Change Employee Information Form

There can only be one root element for a WebCenter Sites page (that is, an entry in the `SiteCatalog` table). This section shows three versions of the root element for the `QueryEditRowForm` page:

- `QueryEditRowFormXML.xml`

- `QueryEditRowFormJSP.jsp`

- `QueryEditRowFormJAVA.jsp`

QueryEditRowFormXML

This is the code in the XML version of the element:

```
<?xml version="1.0" ?>
<!DOCTYPE FTCS SYSTEM "futuretense_cs.dtd">
<FTCS Version="1.1">
<!-- Documentation/CatalogManager/QueryEditRowFormXML
-->

<SETVAR NAME="errno" VALUE="0"/>
<SETVAR NAME="name" VALUE="Variables.EmployeeName"/>
```

```
<SELECTTO FROM="EmployeeInfo"
WHERE="name"
WHAT="*"
LIST="MatchingEmployees"/>

<IF COND="Variables.errno=0">
<THEN>
<form ACTION="ContentServer" method="post">
<input type="hidden" name="pagename" value="Documentation/CatalogManager/
QueryEditRow"/>
<input type="hidden" name="MatchingID" value="MatchingEmployees.id"
REPLACEALL="MatchingEmployees.id"/>
<TABLE>
<TR>
<TD COLSPAN="100%" ALIGN="CENTER">
<H3>Change Employee Information</H3>
</TD>
</TR>
<TR>
<TD>Employee id number: </TD>
<TD><CSVAR NAME="MatchingEmployees.id"/></TD>
</TR>
<TR>
<TD>Employee name: </TD>
<TD><INPUT type="text" value="MatchingEmployees.name" name="NewEmployeeName"
size="22" maxlength="32" REPLACEALL="MatchingEmployees.name"/></TD>
</TR>
<TR>
<TD>Phone number: </TD>
<TD><INPUT type="text" value="MatchingEmployees.phone" name="NewEmployeePhone"
size="12" maxlength="16" REPLACEALL="MatchingEmployees.phone"/></TD>
</TR>
<TR>
<TD colspan="100%" align="center">
<input type="submit" name="doit" value="Change"/></TD>
</TR>
</TABLE>
</form>
</THEN>
<ELSE>
<P>Could not find this employee.</P>
<CALLELEMENT NAME="Documentation/CatalogManager/SelectNameFormXML"/>
</ELSE>
</IF>
</FTCS>
```

When the employee clicks the **Change** button, the information gathered from the two fields and the name of the QueryEditRow page is sent to the browser. The browser sends the page name and the field information to WebCenter Sites. WebCenter Sites looks up the page name in the SiteCatalog table and then invokes that page entry's root element.

QueryEditRowFormJSP

This is the code in the JSP version of the element:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld" %>
<%@ taglib prefix="ics" uri="futuretense_cs/ics.tld" %>
<%//
// Documentation/CatalogManager/QueryEditRowFormJSP
//%>
```

```
<%@ page import="COM.FutureTense.Interfaces.*" %>
<%@ page import="COM.FutureTense.Util.ftMessage"%>
<%@ page import="COM.FutureTense.Util.ftErrors" %>
<cs:ftcs>

<ics:setvar name="errno" value="0"/>
<ics:setvar name="name" value='<%=ics.GetVar("EmployeeName")%>'/>
<ics:selectto table="EmployeeInfo"
where="name"
what="*"
listname="MatchingEmployees"/>

<ics:if condition='<%=ics.GetVar("errno").equals("0")%>'>
<ics:then>
<form action="ContentServer" method="post">
<input type="hidden" name="pagename" value="Documentation/CatalogManager/
QueryEditRow"/>
<input type="hidden" name="MatchingID" value="<ics:listget
listname='MatchingEmployees' fieldname='id'/>"/>
<TABLE>
<TR>
<TD COLSPAN="100%" ALIGN="CENTER">
<H3>Change Employee Information</H3>
</TD>
</TR>
<TR>
<TD>Employee id number: </TD>
<TD><ics:listget listname='MatchingEmployees
fieldname='id'/></TD>
</TR>
<TR>
<TD>Employee name: </TD>
<TD><INPUT type="text" value="<ics:listget
listname='MatchingEmployees' fieldname='name'/>"
name="NewEmployeeName" size="22" maxlength="32"/></TD>
</TR>
<TR>
<TD>Phone number: </TD>
<TD><INPUT type="text" value="<ics:listget
listname='MatchingEmployees' fieldname='phone'/>"
name="NewEmployeePhone" size="12" maxlength="16"/>
</TD>
</TR>
<TR>
<TD colspan="100%" align="center">
<input type="submit" name="doit" value="Change"/></TD>
</TR>
</TABLE>
</form>
</ics:then>
<ics:else>
<P>Could not find this employee.</P>
<ics:callelement element="Documentation/CatalogManager/
SelectNameForm"/>
</ics:else>
</ics:if>

</cs:ftcs>
```

When the employee clicks the **Change** button, the information gathered from the two fields and the name of the `QueryEditRow` page is sent to the browser. The browser

sends the page name and the field information to WebCenter Sites. WebCenter Sites looks up the page name in the `SiteCatalog` table and then invokes that page entry's root element.

QueryEditRowFormJAVA

This is the code in the Java version of the element:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld" %>
<%//
// Documentation/CatalogManager/QueryEditRowFormJAVA
//%>
<%@ page import="COM.FutureTense.Interfaces.*" %>
<%@ page import="COM.FutureTense.Util.ftMessage"%>
<%@ page import="COM.FutureTense.Util.ftErrors" %>
<cs:ftcs>

<!-- user code here -->
<%
ics.SetVar("errno","0");
ics.SetVar("name",ics.GetVar("EmployeeName"));
StringBuffer errstr = new StringBuffer();
IList matchingEmployees = ics.SelectTo("EmployeeInfo",// tablename
*", // what
"name", // where
"name", // orderby
1, // limit
null, // ics list name
true, // cache?
errstr); // error StringBuffer

if ("0".equals(ics.GetVar("errno")) && matchingEmployees!=null &&
matchingEmployees.hasData())
{
%>
<form action="ContentServer" method="post">
<input type="hidden" name="pagename" value="Documentation/CatalogManager/
QueryEditRow"/>
<%
String id = matchingEmployees.getValue("id");
String name = matchingEmployees.getValue("name");
String phone = matchingEmployees.getValue("phone");
%>
<input type="hidden" name="MatchingID" value="<%=id%>"/>
<TABLE>
<TR>
<TD COLSPAN="100%" ALIGN="CENTER">
<H3>Change Employee Information</H3>
</TD>
</TR>
<TR>
<TD>Employee id number: </TD>
<TD><%=id%></TD>
</TR>
<TR>
<TD>Employee name: </TD>
<TD><INPUT type="text" value="<%=name%>" name="NewEmployeeName" size="22"
maxlength="32"/></TD>
</TR>
<TR>
<TD>Phone number: </TD>
```

```
<TD><INPUT type="text" value="<%=phone%>" name="NewEmployeePhone" size="12"
maxlength="16"/></TD>
</TR>
<TR>
<TD colspan="100%" align="center">
<input type="submit" name="doit" value="Change"/></TD>
</TR>
</TABLE>
</form>
<%
}
else
{
%><P>Could not find this employee.</P>
<%
ics.CallElement("Documentation/CatalogManager/SelectNameForm",null);
}
%>
</cs:ftcs>
```

When the employee clicks the **Change** button, the information gathered from the two
fields and the name of the `QueryEditRow` page is sent to the browser. The browser
sends the page name and the field information to WebCenter Sites. WebCenter Sites
looks up the page name in the `SiteCatalog` table and then invokes that page entry's
root element.

12.2.3.3 The Root Element for the QueryEditRow Page

The root element for the `QueryEditRow` page writes the information that the
employee entered into the **Employee Name** and **Phone number** fields and updates the
row in the database.

There can only be one root element for a WebCenter Sites page (that is, an entry in the
`SiteCatalog` table). This section shows three versions of the root element for the
`QueryEditRow` page:

- `QueryEditRowXML.xml`

- `QueryEditRowJSP.jsp`

- `QueryEditRowJAVA.jsp`

QueryEditRowXML

This is the code in the XML version of the element:

```
<?xml version="1.0" ?>
<!DOCTYPE FTCS SYSTEM "futuretense_cs.dtd">
<FTCS Version="1.1">
<!-- Documentation/CatalogManager/QueryEditRowXML
-->

<SETVAR NAME="errno" VALUE="0"/>

<CATALOGMANAGER>
<ARGUMENT NAME="ftcmd" VALUE="updaterow"/>
<ARGUMENT NAME="tablename" VALUE="EmployeeInfo"/>
<ARGUMENT NAME="id" VALUE="Variables.MatchingID"/>
<ARGUMENT NAME="name" VALUE="Variables.NewEmployeeName"/>
<ARGUMENT NAME="phone" VALUE="Variables.NewEmployeePhone"/>
</CATALOGMANAGER>
```

```
<IF COND="Variables.errno=0">
<THEN>
<P>Successfully updated the database.</P>
</THEN>
<ELSE>
<P>Failed to update the information in the database.</P>
</ELSE>
</IF>
</FTCS>
```

QueryEditRowJSP

This is the code in the JSP version of the element:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld" %>
<%@ taglib prefix="ics" uri="futuretense_cs/ics.tld" %>
<%//
// Documentation/CatalogManager/QueryEditRowJSP
//%>
<%@ page import="COM.FutureTense.Interfaces.*" %>
<%@ page import="COM.FutureTense.Util.ftMessage"%>
<%@ page import="COM.FutureTense.Util.ftErrors" %>
<cs:ftcs>

<ics:setvar name="errno" value="0"/>

<ics:catalogmanager>
<ics:argument name="ftcmd" value="updaterow"/>
<ics:argument name="tablename" value="EmployeeInfo"/>
<ics:argument name="id" value="<%=ics.GetVar("MatchingID")%>"/>
<ics:argument name="name"
value='<%=ics.GetVar("NewEmployeeName")%>'/>
<ics:argument name="phone"
value='<%=ics.GetVar("NewEmployeePhone")%>'/>
</ics:catalogmanager>

<ics:if condition='<%=ics.GetVar("errno").equals("0")%>'>
<ics:then>
<P>Successfully updated the database.</P>
</ics:then>
<ics:else>
<p>failed to update the information in the database. errno=<ics:getvar
name='errno'/></p>
</ics:else>
</ics:if>

</cs:ftcs>
```

QueryEditRowJAVA

This is the code in the Java version of the element:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld" %>
<%//
// Documentation/CatalogManager/QueryEditRowJAVA
//%>
<%@ page import="COM.FutureTense.Interfaces.*" %>
<%@ page import="COM.FutureTense.Util.ftMessage"%>
<%@ page import="COM.FutureTense.Util.ftErrors" %>
<cs:ftcs>
```

```
<!-- user code here -->
<%
ics.SetVar("errno","0");

FTValList args = new FTValList();
args.put("ftcmd","updaterow");
args.put("tablename","EmployeeInfo");
args.put("id",ics.GetVar("MatchingID"));
args.put("name",ics.GetVar("NewEmployeeName"));
args.put("phone",ics.GetVar("NewEmployeePhone"));

ics.CatalogManager(args);

if("0".equals(ics.GetVar("errno")))
{
%><P>Successfully updated the database.</P><%
}
else
{
%><p>failed to update the information in the database. errno=<ics:getvar
name='errno'/></p><%
}
%>
</cs:ftcs>
```

12.2.4 How To Query a Table with an Embedded SQL Statement

The following example shows another method of searching for a name in a table. This example also searches the fictitious `EmployeeInfo` table, returning the rows that match the string supplied by a user. But, this time the code uses a SQL query rather than a `SELECTTO` statement.

This section presents code from the following elements:

- `QueryInlineSQLForm`, an XML element that displays a form that requests a movie title.

- Three versions of the `QueryInlineSQL` element (XML, JSP, and Java), an element that searches the `EmployeeInfo` table for names that contain the string entered by the user in the preceding form.

12.2.4.1 QueryInlineSQLForm

The `QueryInlineSQL` element displays a simple form that requests the name for which to search the `EmployeeInfo` table. This is the code:

```
<?xml version="1.0" ?>
<!DOCTYPE FTCS SYSTEM "futuretense_cs.dtd">
<FTCS Version="1.1">
<!-- Documentation/CatalogManager/QueryInlineSQLForm
-->
<form ACTION="ContentServer" method="post">
<input type="hidden" name="pagename" value="Documentation/CatalogManager/
QueryInlineSQL"/>

<table>
<tr>
<td>Employee Name:</td>
<td><input type="text" value="Foo,Bar" name="EmployeeName" size="22"
maxlength="32"/></td>
</tr>
```

```
<tr>
<td colspan="2"><input type="submit" name="submit" value="submit"/></td>
</tr>
</table>

</form>
</FTCS>
```

When the user clicks the **Submit** button, the information gathered in the **Employee name** field and the name of the QueryInlineSQL page is sent to the browser. The browser sends the page name of the QueryInlineSQL page to WebCenter Sites. WebCenter Sites looks up the page name in the SiteCatalog table and then invokes that page entry's root element.

12.2.4.2 The Root Element for the QueryInlineSQL Page

The root element for the QueryInlineSQL page executes an inline SQL statement that searches the EmployeeInfo table for entries that match the string sent to it from the QueryInlineSQLForm element. There can only be one root element for a WebCenter Sites page (that is, an entry in the SiteCatalog table). This section shows three versions of the root element for the QueryInlineSQL page:

- QueryInlineSQLXML.xml: Uses the EXECSQL XML tag to create the SQL query.

- QueryInlineSQLJSP.jsp: Uses the ics:sql JSP tag to create the SQL query.

- QueryInlineSQLJAVA.jsp: Uses the ics.CallSQL Java method to create the SQL query.

QueryInlineSQLXML

This is the code in the XML version of the element:

```
<!DOCTYPE FTCS SYSTEM "futuretense_cs.dtd">
<FTCS Version="1.1">
<!-- Documentation/CatalogManager/QueryInlineSQLXML
-->

<SETVAR NAME="tablename" VALUE="EmployeeInfo"/>

<SQLEXP OUTSTR="MySQLExpression"
TYPE="OR"
VERB="LIKE"
STR="Variables.EmployeeName"
COLNAME="name"/>

<EXECSQL
SQL="SELECT id,name,phone FROM Variables.tablename WHERE Variables.MySQLExpression"
LIST="ReturnedList"
LIMIT="5"/>

<table border="1" bgcolor="99ccff">
<tr>
<th>id</th>
<th>name</th>
<th>phone</th>
</tr>

<LOOP LIST="ReturnedList">
<tr>
```

```
<td><CSVAR NAME="ReturnedList.id"/></td>
<td><CSVAR NAME="ReturnedList.name"/></td>
<td><CSVAR NAME="ReturnedList.phone"/></td>
</tr>
</LOOP>
</table>

</FTCS>
```

Notice that the SQL statement is not actually embedded in the EXECSQL tag. Instead, a preceding SQLEXP tag creates a SQL expression which is passed as an argument to the EXECSQL call. The EXECSQL tag performs the search and returns the results to the list variable named ReturnedList. Also notice that the first line of code in the body of the element creates a variable named tablename and sets the value to EmployeeInfo, the name of the table that is being queried. This enables CatalogManager to cache the resultset against the correct table.

QueryInlineSQLJSP

This is the code in the JSP version of the element:

```
<?xml version="1.0" ?>
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld" %>
<%@ taglib prefix="ics" uri="futuretense_cs/ics.tld" %>
<%//
// Documentation/CatalogManager/QueryInlineSQLJSP
//%>
<%@ page import="COM.FutureTense.Interfaces.*" %>
<%@ page import="COM.FutureTense.Util.ftMessage"%>
<%@ page import="COM.FutureTense.Util.ftErrors" %>
<cs:ftcs>

<!-- user code here -->
<ics:setvar name="tablename" value="EmployeeInfo"/>

<%
// no ics:sqlexp tag, must do in Java
String sqlexp =
ics.SQLExp("EmployeeInfo","OR","LIKE",ics.GetVar("EmployeeName"),"name");
String sql = "SELECT id,name,phone FROM "+ics.GetVar("tablename")+" WHERE "+sqlexp;
%>
<ics:sqltable='<%=ics.GetVar("tablename")%>'
sql='<%=sql%>'
listname="ReturnedList"
limit="5"/>

<table border="1" bgcolor="99ccff">
<tr>
<th>id</th>
<th>name</th>
<th>phone</th>
</tr>

<ics:listloop listname="ReturnedList">
<tr>
<td><ics:listget listname="ReturnedList" fieldname="id"/></td>
<td><ics:listget listname="ReturnedList" fieldname="name"/></td>
<td><ics:listget listname="ReturnedList" fieldname="phone"/></td>
</tr>
</ics:listloop>
```

```
</table>

</cs:ftcs>
```

Notice that the SQL statement is not actually embedded in the `ics:sql` tag. Instead, a preceding Java expression creates a SQL expression that is passed as an argument to the `ics:sql` call. (The code example uses Java because there is no JSP equivalent of the `SQLEXP` tag.) The `ics:sql` tag performs the search and returns the results to the list variable named `ReturnedList`. Also notice that the first line of code in the body of the element creates a variable named `tablename` and sets the value to `EmployeeInfo`, the name of the table that is being queried. This enables CatalogManager to cache the resultset against the correct table.

QueryInlineSQLJava

This is the code in the Java version of the element:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld" %>
<%//
// Documentation/CatalogManager/QueryInlineSQLJAVA
//%>
<%@ page import="COM.FutureTense.Interfaces.*" %>
<%@ page import="COM.FutureTense.Util.ftMessage"%>
<%@ page import="COM.FutureTense.Util.ftErrors" %>
<cs:ftcs>

<%

ics.SetVar("tablename","EmployeeInfo");

String sqlexp =
ics.SQLExp(ics.GetVar("tablename"),"OR","LIKE",ics.GetVar("EmployeeName"),"name");
String sql = "SELECT id,name,phone FROM "+ics.GetVar("tablename")+" WHERE "+sqlexp;
StringBuffer errstr = new StringBuffer();

IList list = ics.SQL(ics.GetVar("tablename"),sql,null,5,true,errstr);

%>

<table border="1" bgcolor="99ccff">
<tr>
<th>id</th>
<th>name</th>
<th>phone</th>
</tr>

<%

while (true)
{
%>
<tr>
<td><%=list.getValue("id")%></td>
<td><%=list.getValue("name")%></td>
<td><%=list.getValue("phone")%></td>
</tr>
<%
if (list.currentRow() == list.numRows())
break;
list.moveTo(list.currentRow()+1);
```

```
}
%>

</table>
</cs:ftcs>
```

Notice that the SQL statement is not actually embedded in the `ics.SQL` statement. Instead, a preceding `ics.SQLExp` statement creates a SQL expression which is passed as an argument to the `EXECSQL` call. The `ics.SQL` statement performs the search and returns the results to the list variable named `ReturnedList`. Also notice that this code also creates a variable named `tablename` and sets the value to `EmployeeInfo` (the name of the table that is being queried), before the code for the query. This enables CatalogManager to cache the resultset against the correct table.

12.3 Consideration About Deleting Non-Asset Tables

All you need to do before deleting a non-asset table is disable revision-tracking.

To delete a non-asset table which is being revision tracked, first disable revision tracking for the table. Otherwise, the table can't be removed.

Part III

Developing a Website

Get introduced to WebCenter Sites tags and Java methods. These tags and methods let you store pagelets in WebCenter Sites caches, maintain those caches for your visitors to always see updated information, and these tags and methods let you do much more. Know how you can create CSElement, Template, and SiteEntry asset types. Familiarize yourself with templates and wrappers, too.

Learn about coding templates for in-context editing and presentation editing. Procedures are available to help you create a page, query, and collection assets that implement the functionality of your online site, to help you implement Future Site Preview functionality, and to help you code the elements that you make for your template and CSElement assets. Take a look at the examples for using basic assets. And, if you're planning to configure multilingual support for your site, detailed information is available to help you do that.

- Website Development with the MVC Framework and APIs

- Developing a Server-Side Website

- Developing a Client-Side Website

- Website Development with Tag Technologies

- About Sessions and Cookies

- Creating Template, CSElement, and SiteEntry Assets

- Creating Templates and Wrappers

- Coding Templates for In-Context and Presentation Editing

- Creating Collection Assets, Query Assets, and Page Assets

- Best Practices for Creating Future Site Preview Assets and Templates

- Coding Elements for Templates and CSElements

- Template Element Examples for Basic Assets

- Configuring Sites for Multilingual Support

13

Website Development with the MVC Framework and APIs

Oracle WebCenter Sites provides a Model-View-Controller (MVC) framework and Java APIs for developing server-side websites, as well as REST APIs for building websites rendered on the client side. With these technologies you can develop flexible and scalable websites.

Topics:

- Server-Side and Client-Side Development Methodologies

- Server-Side MVC Framework

- Pages, Pagelets, and Elements

- Template and CSElement Assets

- Page Assets and Site Navigation

- Date-Based Preview

- Multilingual Support

- Caching in the MVC Framework

- Server-Side Java APIs

- REST APIs

- Sample Websites

13.1 Server-Side and Client-Side Development Methodologies

Oracle WebCenter Sites provides tools for building server-side rendering of content on web pages, client-side rendering, or a combination of both. Each methodology offers unique advantages, depending on your website design and the content you want to render. Both methodologies enable you to prototype and develop websites in an easy-to-use, standardized, and compliant way, for rapid and productive development.

- The *server-side Model-View-Controller (MVC) framework* cleanly separates presentation from business logic, to support robust server-side scripting.

  Java developers can write business-logic code in the `Controller` using Groovy and write `Templates` in JSP, while web developers can create presentation elements in views using HTML and JavaScript, without interfering with each other's work. This MVC framework is fully integrated with WebCenter Sites caching, asset dependency, and lifecycle management features.

To streamline coding, the framework includes simplified server-side Java APIs. The out-of-the-box Java APIs provide convenient interfaces for common activities in building a website, such as these tasks:

– Reading asset data

– Painting breadcrumbs for pages

– Accessing form values

– Performing validation

– Using a `Controller` as a REST endpoint

See Server-Side Java APIs.

Note:

The Java APIs in conjunction with the MVC framework replace the WebCenter Sites JSP tag technology for elements and templates, described in Website Development with Tag Technologies However, WebCenter Sites tags remain supported.

- *REST APIs* enable you to build a website rendered on the client side, using technologies that execute entirely in the browser client. Such websites include, but are not limited to, single-page applications.

  A standalone sample website demonstrating the WebCenter Sites REST API capabilities is shipped as part of the WebCenter Sites installation download. See the instructions in the `wcsites` directory on how to configure this sample reference implementation.

  Rendering on client-side websites is particularly useful for highly interactive applications (or portions of websites) and mobile sites. Client-side rendering, in general, delegates the markup generation to client-side libraries while limiting the interactions between the client and server to just data elements, usually in the form of JSON (JavaScript Object Notation). As users navigate from page to page in such a website, requests are made to the server for specific data components while the JavaScript to generate the markup stays loaded in the browser.

  REST APIs get data components to the client efficiently, in JSON format. These APIs mirror features of server-side Java APIs in general. More important, these services have additional features, such as aggregation, for the special needs of client-side rendering.

  The REST APIs address common usage patterns such as these:

  – Accessing navigation data

  – Accessing aggregated asset data

    For example, the APIs can exclude certain fields, use wildcards, or build lists.

  Depending on the needs of visitors to your website, you might decide on an entirely server-side or client-side methodology to build it. Or you might opt for a mix of both techniques. Contributors to pages rendered on the server side can compose them in web mode, using the Oracle WebCenter Sites: Contributor interface, but web-mode editing is not available for elements of a page that are rendered on the client side.

See REST APIs.

13.2 Server-Side MVC Framework

The MVC design pattern involves the separation of model, view, and `Controller`. Model represents the data, view represents the presentation, and `Controller` links the model and the view. Clear separation makes the code readable and easily modifiable, and it's easy to accommodate technological changes.

- The *model* represents the data.

  In WebCenter Sites, a model is a map of key-value pairs. Asset data as well as any data determined in business logic would be part of the model.

- The *view* represents the presentation.

  In WebCenter Sites, a view is a `Template` or `SiteEntry` object that renders to a browser. The view contains no business logic, and it renders the model data in an interface for users to view and modify the data. It also sends user actions (such as visitor clicks) to the `Controller`.

- The *Controller* links the model and the view. Performing business logic on underlying data, the `Controller` puts together a model in a way that can be used in a view.

  In WebCenter Sites, a `Controller` is a standard asset instance. A presentation asset (`Template` or `SiteEntry` asset type) can link to a `Controller`. This link is optional. When it is present, WebCenter Sites invokes the linked `Controller` before invoking the view. As with any asset type, assets of type `Controller` can be approved and published.

  `Controllers` can extend and import other `Controllers`. A `BaseController` provides convenient accessors, debugging, and much more. In addition, you can use Java APIs in a `Controller`, as described in Server-Side Java APIs.

  For information about using a `Controller` as a REST endpoint, go to the Developer's Samples Website and choose **REST Endpoint** from the **Advanced Topics** menu.

Note:

If a view (`Template` or `SiteEntry` asset) is not associated with a `Controller`, WebCenter Sites treats it as a standalone view. Such a view falls back to the legacy view-based rendering of WebCenter Sites, described in Using Asset API: Tutorial. WebCenter Sites remains fully backwards compatible, allowing you the option of transitioning incrementally to the MVC framework.

Note:

If a `Controller` extends and imports another `Controller`, editing of the extended `Controller` does not invalidate the cache for the extending `Controller` by default. To invalidate the cache for the extending `Controller`, or `Controllers`, either edit and save each extending `Controller` from the Admin interface, or use the Controller Asset Utility in System Tools to force-compile all dependent `Controllers`.

The following topics describe parts of the server-side MVC framework:

- Developer's Samples Website

- WebCenter Sites MVC Framework Overview

- Controllers

- Views

13.2.1 Developer's Samples Website

The Developer's Samples website provides an overview of the new MVC framework, examples for getting started, server-side Java API examples, and advanced topics such as Annotations, Select Grammar, Custom Beans, Form Validation, REST endpoints, Fragments, and Watchers.

This reference website is based on the Samples Site in the Contributor interface.

To access the Developer's Samples website, enter the following URL, with the host name and port number for your installation:

```
http://hostname:port/sites/samples/overview
```

13.2.2 WebCenter Sites MVC Framework Overview

This figure shows the anatomy of a request as it passes through the MVC framework to produce a response.

Figure 13-1   MVC Framework Process for Server-Side Rendering
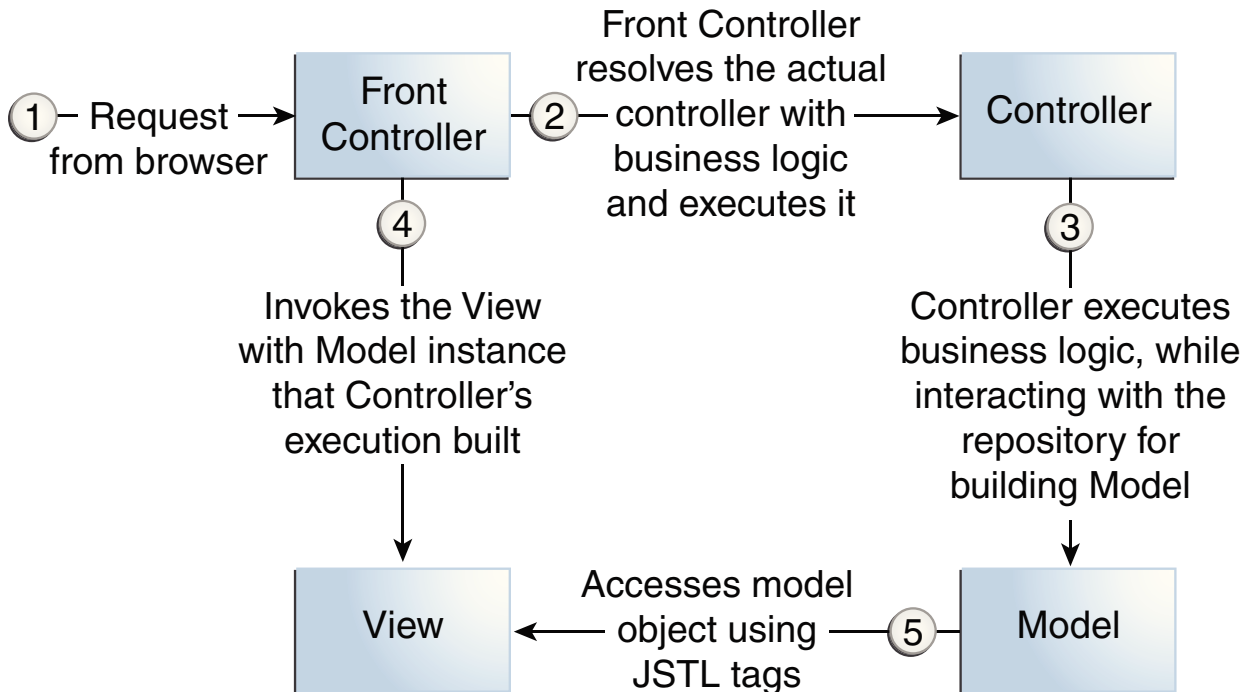
In WebCenter Sites, the MVC framework includes a front `Controller`. Oracle WebCenter Content Server or Satellite Server servlets act as the front `Controller` and handle various dispatching and caching services.

When a request from a client reaches the front `Controller`, it resolves the actual `Controller` instance associated with the `Template` that contains business logic for

the request, depending on the request URL. The front `Controller` does this by first locating the view element (a `Template` asset, for instance) and then looking up a `Controller` instance associated with the view (if the `Template` is linked with a `Controller`).

After discovering the `Controller` instance that contains business logic for the page, the front `Controller` executes it. The result of executing a `Controller` is the creation of a model instance.

The front `Controller` then dispatches the request to the view with the model instance placed in the request scope. The view (a JSP) can now access the model instance using JSTL tags and present it in response back to the browser.

The same process repeats for *pagelets* (parts of a web page). Pagelets with their `Controller` objects become reusable components that can be reused across many pages.

In addition to managing your content, WebCenter Sites handles many useful tasks for you, such as storing web pages and pagelets in WebCenter Sites caches and maintaining those caches so that visitors to your website never see an outdated page. You can use WebCenter Sites Java methods to do this.

Various *element assets* and *Template assets* compose a WebCenter Sites page. Element assets include blocks of code that can retrieve the content of your pages from the WebCenter Sites database or perform other tasks, such as deleting outdated items from the database. `Template` assets are generally used to format the content of your web pages. Elements and `Templates` can be written in JSP.

13.2.3 Controllers

`Controllers` contain business logic in code authored by developers. The programming language for writing such code in WebCenter Sites 12*c* is Groovy, a language similar to Java. Groovy is built on Java and interoperates easily with Java libraries.

Business logic in `Controllers` is usually specific to a website implementation. However, a large number of pages or pagelets typically need to perform some very common operations. For example, a `Controller` for a page might determine a list of assets to show for the view and show each asset in a pagelet. Conceivably, custom business logic might be needed to come up with a list of assets (such as a query of certain articles). In this case, the only `Controller` logic necessary is searching for assets. This logic can be the same for the entire website.

WebCenter Sites ships with a set of `Controllers` already built with common patterns of use, like navigation for reading assets. Oracle encourages developers to use the following out-of-the-box `Controllers` where applicable:

- `AssetController`

- `NavigationController`

- `TableController`

- `SegmentsController`

- `ProfileBasedRecommendationController`

- `SegmentBasedRecommendationController`

Parts of a page might have custom `Controllers` as well as views with additional `Controllers`. You can extend `Controllers`.

13.2.4 Views

In WebCenter Sites, `Template` and `SiteEntry` assets represent views. Each `Template` and `SiteEntry` asset can be linked to a `Controller` instance.

When you build views using the MVC framework, Oracle recommends that you use JSP as the scripting language for building the views and JSTL to access data in the model. While WebCenter Sites tags for accessing asset data continue to work when used in `Template` assets, Oracle recommends that developers use Java APIs in the `Controllers` instead, and use view code only for managing presentation. This separates logic from presentation. See Server-Side Java APIs.

13.3 Pages, Pagelets, and Elements

Template and SiteEntry assets can be both pages and pagelets, but CSElement is always an element. Assets are displayed in pages or pagelets using templates and elements.

In the WebCenter Sites context, an online page is the composition of several components into a viewable, final output. Creating that output is called *rendering*. Making either that output or the content that is to be rendered available to the visitors on your public site is called *publishing*.

WebCenter Sites renders pages by executing the code associated with page names. The name of a page is passed to WebCenter Sites from a browser, and WebCenter Sites invokes the code associated with that page name. The code is actually in a named file, a separate chunk of code called an element.

The code in your elements identifies assets to display in pages or pagelets, loads the assets, and then passes other page names and element names to WebCenter Sites. When WebCenter Sites invokes an element, all of the code in the element is executed. If there are calls to other elements, those elements are invoked in turn. Then the results, the images, articles, linksets, and so on, including any HTML tags, are rendered into HTML code (or some other output format if your system is configured to do so).

`Template`, `CSElement`, and `SiteEntry` assets represent elements and pagelets as follows:

- A `Template` asset is both an element and a page or pagelet that renders an asset.

- A `CSElement` asset is an element.

- A `SiteEntry` asset is the name of a page or a pagelet.

13.4 Template and CSElement Assets

Here are some points that you need to know about Template and CSElement assets.

- `Template` assets are classified as typed or typeless depending on whether they apply to a single asset type or no asset type.

- If you are using SiteLauncher (to replicate sites or to share `Template` and `CSElement` assets), WebCenter Sites requires element logic to indirectly refer to assets, asset types, attribute names, and `Template` names. To this end, the WebCenter Sites interface introduces the Map screen, and the API introduces the `render:lookup` tag.

Using the Map screen, you assign an alias to each value. You can then hard-code the aliases in the element logic and use the `render:lookup` tag to retrieve the actual values from the aliases at runtime.

- The **Cache Rules** field has been simplified to reduce errors. `Template` developers can choose **Cached**, **Uncached**, or **Advanced**. Choosing **Advanced** allows developers to set caching rules individually for WebCenter Sites and Oracle WebCenter Sites: Satellite Server.

- A new tag, `calltemplate`, was introduced to invoke `Templates` in a way that simplifies the `Template` writing process.

- The **PageCriteria** field has been renamed to **Cache Criteria**. It accepts the following reserved parameters: `c`, `cid`, `context`, `p`, `rendermode`, `site`, `sitepfx`, `ft_ss`, and custom-defined parameters.

  Cache criteria values are stored in the `pagecriteria` column of the `SiteCatalog` table (in previous versions they were stored in the `resargs` columns of the `SiteCatalog` table).

  The **Cache Criteria** field is also used to hold variables that enable the **Extra Parameters** section in the CKEditor and make them available to users, in the **Include asset link** and **Add asset link** dialog boxes. The **Extra Parameters** section provides a way of passing custom parameters (such as image dimensions) to the `Template`. See Working with the CKEditor in *Oracle Fusion Middleware Using Oracle WebCenter Sites*.

- Forms for creating `Template` and `CSElement` assets have been subdivided by tabs, and fields are organized by function on the tabs.

See Creating Template, CSElement, and SiteEntry Assets

13.5 Page Assets and Site Navigation

Page assets store references to other assets and organize them according to a site design. Representing sections of a site, page assets provide a convenient structural organization that resembles a finished website.

Site developers typically create page assets during site design. See Creating Page Assets.

You can associate pages, articles, and other type of assets with page assets and code template elements that format the associated assets.

A `Template` associated with a page asset contains the layout for the rendered page and navigation to other rendered pages.

To put together site navigation for page assets, you can use the WebCenter SitesWebCenter Sites Java APIs or REST APIs. See Server-Side Java APIs and REST APIs.

13.6 Date-Based Preview

Using date-based rendering, content contributors can preview assets in the Contributor interface. If you are interested in building a website that changes based on the date of viewing, then you can use start dates and end dates available in an asset. Such websites can also be previewed for how they would look on a future date by using WebCenter Sites preview tools.

Content creators can specify start dates and end dates on Edit screens in the Contributor interface. They can preview content through the site's user interface and then go to a date in the future and look at how the site would look on that date.

For example, you might want to build a site for a holiday and preview how it will look then.

See Asset Reader, a Java API that provides a convenience method to filter by date.

13.7 Multilingual Support

Users of a multilingual site developed with WebCenter Sites can assign locale (language version) designations to assets and create translations of assets. You can create site-specific delivery rules that determine the assets' language versions to be shown on the online site, and what should happen if the requested language version doesn't exist.

Locale designations in WebCenter Sites are implemented through the concept of dimensions. A dimension is an identifier that differentiates assets that are otherwise semantically identical. A locale (such as `en_US` for US English) is a type of dimension that differentiates two translations of the same content.

Dimensions are represented by assets of the type `Dimension`. This asset type must be enabled by developers on a per-site basis so that users can create `Dimension` assets (of subtype `Locale`) and so that content providers can assign locale designations to assets they want to translate.

> **Note:** Users cannot create translations of assets that have no locale designation assigned.

Each `Dimension` asset represents a locale on the site. For example, an `en_US` `Dimension` asset represents US English, and a `fr_CA` `Dimension` asset represents Canadian French. See Asset Reader, a Java API that gives you the ability to access translated assets.

Publishing content in a given locale requires enabling the locale on the online site. This publishes the `Dimension` asset representing the locale to the delivery system and includes the locale in the site's `Dimension` set.

13.8 Caching in the MVC Framework

Page caching in the MVC Framework functions similar to the `Template` assets coded using JSP tags. In the new framework, business logic code moves to the `Controller` asset, leaving the `Template` asset responsible for presentation aspects only. What gets cached in page caching is still the page or pagelet markup. Business logic associated with a pagelet runs only when the pagelet is uncached.

A `Controller` asset associated with a `Template` is added as a cache dependency, along with all the assets read in the `Controller`'s business logic. Any updates to the `Controller` asset or the assets read as part of the `Controller`'s business logic would invalidate the cache.

The cache criteria arguments in the `Template` asset continue to determine the caching behavior at the pagelet level. For a description of how caching works in WebCenter Sites, see Understanding Page Design and Caching.

13.9 Server-Side Java APIs

To put together a web page, you typically would use various kinds of server-side Java APIs.

WebCenter Sites provides:

- Asset Reader

- Navigation Reader

- Link Builder

- Blob Link Builder

- Searcher

- Recommendation Reader

- Table Reader

Brief descriptions and samples of these APIs follow. The Developer's Samples Website provides additional descriptions and examples.

13.9.1 Asset Reader

The Asset Reader API reads an asset and returns all data or the data you specify. You can use the methods in `AssetReader` to read the asset in a simplified way and return only required and relevant information.

A `Controller` uses these methods in conjunction with each other to build assets that a `Template` can readily use. This example shows a simple `Controller` that can use`AssetReader` to read an asset.

Example 13-1   Asset Reader Controller

```
package oracle.webcenter.sites.controller

import com.fatwire.assetapi.data.*
public class AssetController extends BaseController
{
      @RequiredParams(query="c,cid")
      public void doWork(Map models)
      {
            Map assetMap = newAssetReader()
                               .forAsset(getAssetId())
                               .selectAll(true)
                               .read();
            models.put("asset", assetMap);
      }
```

This `assetData` map is available to a `Template` that chooses `AssetController` as its `Controller`. This example shows what a `Template` that renders the asset would look like:

Example 13-2   Template for Asset Reader

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"
%><%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"
%><cs:ftcs><%-- AVIArticle/myarticle --%>
<table border="1">
    <c:forEach var="a" items="${asset}">
```

```
        <tbody>
            <tr>
                <td><h4>${a.key} : </h4></td>
                <td>${a.value}</td>
            </tr>
            </tbody>
        </c:forEach>
</table>
</cs:ftcs>
```

The `Controller` reads the asset with asset ID `'1328196047443'` and asset type `'AVIArticle'` and puts it in the map `'assetData'`. The `selectAll(true)` method builds the asset with all of its attributes.

You can go to the Developer's Samples Website and choose **Asset Reader** from the **Rendering API** menu for more information.

13.9.2 Navigation Reader

The Navigation Reader API paints breadcrumbs for navigation.

This example shows a sample Navigation `Controller` for an asset.

Example 13-3   Navigation Controller

```
package oracle.webcenter.sites.controller
import com.fatwire.assetapi.data.*
import com.openmarket.xcelerate.asset.*

public class NavigationController extends BaseController
{
        @RequiredParams(query="c,cid")
        public void doWork(Map models)
        {
                Map assetMap = newNavigationReader()
                    .forAsset(getAssetId())
                    .read();
                models.put("asset", assetMap);
        }
}
```

This example shows a sample view (Template) for the Navigation `Controller`.

Example 13-4   View for Navigation Controller

```
<%@ taglib prefix="render" uri="futuretense_cs/ftcs1_0.tld"

%><%-- Page/avipage --%>

        Asset Name : ${asset.name}

        Asset Id : ${asset.name}

        Asset Parent : ${asset.parents[0].name}
```

The `newNavigationReader()` method in `BaseController` returns a Navigation Reader. The preceding sample `Controller` fetches the immediate parent of the asset in the site plan. For example, the hierarchy of the nontouch Surfing page in avisports is `NonTouch/Home/Surfing`, and the preceding `Template` prints the following text:

```
Asset Name : Surfing
Asset Id : Surfing
Asset Parent : Home
```

You can go to the Developer's Samples Website and choose **Navigation Reader** from the **Rendering API** menu for more information.

13.9.3 Link Builder

The Link Builder API generates a link to an asset.

This example shows a sample `Controller`.

Example 13-5   Link Builder Controller

```
package oracle.webcenter.sites.controller

import com.fatwire.assetapi.data.*
import com.openmarket.xcelerate.asset.*

public class linfo extends BaseController {

        @Override
        public void doWork(Map models) {
        LinkInfo linkInfo = newAssetLinkInfo();
        linkInfo.forAsset(getAssetId());

        Map assetMap = newAssetReader()
            .forAsset(assetID)
            .selectAll(true)
            .addAssetLinkInfo(linkInfo)
            .read();
        models.put("asset", assetMap);
        }
}
```

This example shows a sample `Template` for Link Builder.

Example 13-6   Link Builder Template

```
<%@ taglib prefix="render" uri="futuretense_cs/ftcs1_0.tld"%>

    <cs:ftcs><%-- AVIArticle/myarticle --%>
        <table border=1>
            <tr>
                <td><h4> Asset Name : </td>
                <td></h4> ${asset.name} </td>
            </tr>
            <tr>
                <td><h4>Link: </td>
                <td?</h4><a href="${asset._link_}"> ${asset._link_}</a></td>
            </tr>
        </table>
    </cd:ftcs>
```

The preceding code samples show a simple `Controller` and `Template` for `LinkInfo`. This code generates a link for an `Article` with the asset ID '1328196047443'. The asset map contains a key, `_link_`. whose value is the default URL for the asset.

In the `Template`, `${asset._link_}` should fetch the default URL of the asset. The link would look like this:

```
http://localhost:8080/cs/avi/avisection/baseball.html
```

For more information, go to the Developer's Samples Website and choose **Link Builder** from the **Rendering API** menu.

13.9.4 Blob Link Builder

The Blob Link Builder API generates a link to a blob. If you want the blob links to be displayed when you read an asset, you can use the `addBlobLinkInfo(BlobLinkInfo blobLinkMaker)` method.

This example shows a sample `Controller`.

Example 13-7   Blob Link Builder Controller

```
package oracle.webcenter.sites.controller

import com.fatwire.assetapi.data.*
import com.openmarket.xcelerate.asset.*

public class bloblink extends BaseController
{
    public void doWork(Map models)
    {

      def mapParams = [:]
      mapParams.put("blobheadername1","Content-Type")
      mapParams.put("blobheadervalue1","image/gif")
      mapParams.put("blobheadername2","Cache-Control")
      mapParams.put("blobheadervalue2","no-cache")

      BlobLinkInfo blobLink = newBlobLinkInfo();
      blobLink.parameters(mapParams)


      Map assetMap = newAssetReader().forAsset(getAssetId())
            .select("Assoc_Named_Manual.*")
            .addBlobLinkInfo(blobLink)
            .read();
        models.put("asset", assetMap);
    }
}
```

This example shows a sample `Template` for Blob Link Builder.

Example 13-8   Blob Link Builder Template

```
<cs:ftcs><%-- Product_C/ptemplate --%>
    <table border=1>
    <tr>
    <td><h4>Link1: </td>
    <td></h4><a href='${asset["Assoc_Named_Manual.FSIIDocumentFile_bloblink_"]}'>
    ${asset["Assoc_Named_Manual.FSIIDocumentFile_bloblink_"]}</a></td>
    </tr>
    <tr>
    <td><h4>Link2: </td>
    <td></h4><a href='${asset["Assoc_Named_Manual.FSIIHtmlFile_bloblink_"]}'>
    ${asset["Assoc_Named_Manual.FSIIHtmlFile_bloblink_"]}</a></td>
    </tr>
    </table>
    </cs:ftcs>
```

The preceding code prints the link to any blob assets that are associated with Product_ C with the asset ID 1114083739596. This asset has a Document_C association, which contains two blobs, FSIIDocumentFile and FSIIHtmlFile. The code displays links to those blobs:

```
/sites/BlobServer?blobkey=id&blobnocache=true&blobwhere=1114725604909&blobcol=urldata
&blobtable=MungoBlobs&Content-Type=image/gif&Cache-Control&no-cache
```

```
/sites/BlobServer?blobkey=id&blobnocache=true&blobwhere=1114725604910&blobcol=urldata
&blobtable=MungoBlobs&Content-Type=image/gif&Cache-Control&no-cache
```

If there is a vanity URL, the code displays it.

You can go to the Developer's Samples Website and choose **Blob Link Builder** from the **Rendering API** menu for more information.

13.9.5 Searcher

The Searcher API builds search queries over assets

Example 13-9   Searcher Controller

This example shows a sample Template for Searcher.

```
package oracle.webcenter.sites.controller

import com.fatwire.assetapi.data.*
import com.openmarket.xcelerate.asset.*

public class SearchController extends BaseController
{
    @RequiredParams(query="c,cid")
    public void doWork(Map models)
    {
        def results = newSearcher().searchFor("audio")
        models.put("results",results)
    }
}
```

This example shows a sample Template for Searcher.

Example 13-10   Searcher Template

```
<%@ taglib prefix="render" uri="futuretense_cs/ftcs1_0.tld"%>

<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"
%><cs:ftcs><%-- Product_C/ptemplate --%>
    <table>
        <c:forEach var="result" items="${results}" >
            <tr>
                <td>
                    ${result.id}
                </td>
                <td>
                    ${result.name}
                </td>
            </tr>
        </c:forEach>
    </table>
</cs:ftcs>
```

The newSearcher() method in BaseController returns a Searcher. You can apply one or more methods in the Searcher class to this Searcher. The preceding Controller gathers all the assets that contain the word "audio". To find all the assets that contain the word and that belong to a particular site, you could use the inSite(String site) method:

```
def results = newSearcher().inSite("FirstSiteII").searchFor("audio")
```

You can go to the Developer's Samples Website and choose **Searcher** from the **Rendering API** menu for more information.

13.9.6 Recommendation Reader

The Recommendation Reader API reads recommendations for a visitor based on segmentation of visitors by attributes, such as gender or hobbies. One or more assets are recommended to visitors in a Segment.

To read all the Segments a particular visitor might fall under, you can use the readSegments() method and a Template for Recommendation Reader, as the example shows.

Example 13-11   Method to Read Segments

```
 Controller Logic:
.
package oracle.webcenter.sites.controller
.
import com.fatwire.assetapi.data.*
import com.openmarket.xcelerate.asset.*
import com.fatwire.assetapi.fragment.*
import com.fatwire.assetapi.common.AssetAccessException;
.
public class RecommendationController extends BaseController
{
    @Override
@RequiredParams(query = "segments,recommendation,sitename")
protected void doWork(Map models) {
String segments = variables.segments;
String recommendation = variables.recommendation;
String sitename = variables.sitename;
try {
List<Map> recommendations = newRecommendationReader()
.forSite(sitename)
.forSegments(segments)
.readRecommendations(recommendation);
models.put("segments", segments);
models.put("recommendation", recommendation);
models.put("sitename", sitename);
models.put("recommendations", recommendations);
} catch (AssetAccessException e) {
e.printStackTrace();
}
}
}
.
Template Logic:
.
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"
%><%@ taglib prefix="ics" uri="futuretense_cs/ics.tld"
%><%@ taglib prefix="render" uri="futuretense_cs/render.tld"
%><%@ taglib prefix="fragment" uri="futuretense_cs/fragment.tld"
```

```
%><%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"
%>
<cs:ftcs><%-- /RecommendationTemplate --%>
<%-- Record dependencies for the Template --%>
<ics:if condition='<%=ics.GetVar("tid")!=null%>'><ics:then><render:logdep
cid='<%=ics.GetVar("tid")%>' c="Template"/></ics:then></ics:if>
.
Segments : ${segments}
Recommendation : ${recommendation}
Site Name : ${sitename}
Recommendations :
.
    <table>
      <c:forEach var="singleRecommendation" items="${recommendations}" >
        <tr>
            <td>
                ${singleRecommendation}
            </td>
        </tr>
      </c:forEach>
    </table>
.
</cs:ftcs>
```

The preceding code sample displays all the recommended assets for the given recommendation, in the given site, for the current segments.

You can go to the Developer's Samples Website and choose **Recommendation Reader** from the **Rendering API** menu for more information.

13.9.7 Table Reader

The Table Reader API queries tables managed by WebCenter Sites.

This example shows a sample `Controller` for Table Reader.

Example 13-12   Table Reader Controller

```
package oracle.webcenter.sites.controller

import com.fatwire.assetapi.data.*
import com.openmarket.xcelerate.asset.*
import oracle.fatwire.api.TableReader;

public class TableReader extends BaseController
{
    public void doWork(Map models)
    {
        TableReader tr = newTableReader()
        def myResult = tr.from("AVIArticle").execute()
        models.put("asset", myResult)
    }
}
```

The new `TableReader()` method is a factory method in `BaseController`. The preceding code sample fetches all the data from the `AVIArticle` table and puts it in a map named `asset`.

You can specify a `SELECT` parameter with the `select(String select)` method:

```
def myResult = tr.select("name").from("AVIArticle").execute()
```

This code returns all the values in the **name** column of the **AVIArticle** table.

You can go to the Developer's Samples Website and choose **Table Reader** from the **Rendering API** menu for more information.

13.10 REST APIs

WebCenter Sites provides REST APIs for accessing content in an aggregate fashion. The APIs have self-explanatory URLs. The search APIs support large metadata to provide a variety of features. For instance, you can get a customized list of assets of a particular type with specific values in the chosen fields.

See the *Oracle Fusion Middleware Aggregate REST API Reference for Oracle WebCenter Sites*.

13.11 Sample Websites

You can use the WebCenter Sites sample websites to familiarize yourself with the WebCenter Sites framework.

- Samples Site

  See Developer's Samples Website.

- avisports sample site

  This site illustrates features in the WebCenter Sites Contributor interface such as creating and editing assets in Form Mode and Web Mode. In addition, avisports provides developers with sample `Templates` that are coded to render an asset's Create or Edit view, or both, in Web Mode.

- REST-avisports sample website

  This site has a sample reference implementation of avisports built completely with REST APIs. The functionality aspect of the website is identical to avisports and uses the same asset model and content. This sample site is a good reference point for getting started with website development based on REST APIs.

14

Developing a Server-Side Website

In the Oracle WebCenter Sites Model-View-Controller Framework, to code your organization's data model, you, the Java developer, can write Templates for views in JSP and use Groovy to code business logic in Controllers, streamlined with WebCenter Sites Java APIs. Web developers can add presentation elements to views.

Topics:

- About Developing a Server-Side Website

- Working with the Controller Interface

- Creating a Controller

- Creating a Template

- Setting Up the Home Page

- Adding Site Navigation

14.1 About Developing a Server-Side Website

In the Oracle WebCenter Sites Model-View-Controller (MVC) Framework, the model is your data model. For the view, you create a `Template` or `SiteEntry` object. You can use an out-of-the-box `Controller` or create one to tie the model and view together.

The Oracle WebCenter Sites Model-View-Controller (MVC) Framework is described in Server-Side MVC Framework.

WebCenter Sites provides several `Controllers` already built with common patterns of use, like navigation for reading assets. For information about these out-of-the-box `Controllers`, see Controllers, or go to the Developer's Samples Website and choose **Out of the Box Controllers** from the **Getting Started** menu.

14.2 Working with the Controller Interface

The Controller interface includes the `handleRequest()` method that lets you write an independent WebCenter Sites controller with independent tests.

The interface is as follows:

```
public interface Controller
{

    /**
     * Run the controller with all the information passed in from setters
     * @return a ModelAndView object
     * @throws ControllerException throws this Exception if errors occurred in
processing
     */
```

```
    DependenciesAwareModelAndView handleRequest() throws ControllerException;
}
```

A new WCSController is created to provide a set of default WebCenter Sites-specific functions.

```
public interface WCSController extends Controller
{
    /**
     * setting an ics into the Controller, ICS is needed for retrieving information
related to Controller
     * @param ics the ICS object
     */
    public void setICS( ICS ics );

    /**
     * Get the current device
     * @return the device object
     */
    public Device getDevice();

    /**
     * set the current device
     * @param device the device
     */
    public void setDevice(Device device);

    /**
     * get the query parameters specified for this request. For multi-valued query
parameter, provide the
     * name=value for each parameter value. For eg.
attributes=name&attributes=description&attributes=tag.
     * @return the query parameters map
     */
    public Map<String, List<String>> getQueryParams();

    /**
     * get the headers for this request
     * @return the headers map
     */
    public Map<String, List<String>> getHeaders();

    /**
     * get the template parameters specified in Site for this requested page
     * @return the template parameters map
     */
    public Map<String, String> getTemplateParams();

    /**
     * get the current view
     * @return the view information
     */
    public String getViewMarkup();

    /**
     * get the template map information
     * @return the template map
     */
    public List<Map<String, String>> getTemplateMap();

    /**
```

```
 * get the current session variables for the request
 * @return the session variables
 */
public Map<String, String> getSessionVariables();

/**
 * get the variables that were available in current ICS
 * @return the variables
 */
public Map<String, String> getVariables();

/**
 * get the element catalog parameters specified in Site for this requested page
 * @return the element catalog parameters map
 */
public Map<String, String> getElementCatalogParameters();


/**
 * setting the current view into the Controller
 * @param view the current view markup
 */
void setViewMarkup(String view);

/**
 * setting the the redirect view pagename into the Controller
 * @param redirectViewPagename the redirect view pagename
 */
void setRedirectViewPagename(String redirectViewPagename);

/**
 * Getting the redirect view pagename
 * @return the redreict view pagename
 */
String getRedirectViewPagename();

/**
 * Set all request parameters into Controller
 * @param queryParams the request parameters
 */
void setQueryParams(Map<String, List<String>> queryParams);

/**
 * Set all request headers into the view
 * @param headers the request headers
 */
void setHeaders(Map<String, List<String>> headers);

/**
 * Set all template parameters related to current view into Controller
 * @param templateParams the template parameters (SiteCatalog parameters)
 */
void setTemplateParams(Map<String, String> templateParams);

/**
 * Set all template map information into the Controller
 * @param templateMap the list of template map information for current page
 */
void setTemplateMap(List<Map<String, String>> templateMap);

/**
```

```
 * Set all ics session variables into the view
 * @param sessionVariables the current ics session variables
 */
void setSessionVariables( Map<String, String> sessionVariables );

/**
 * Set all ics variables into the view
 * @param variables the current ics variables
 */
void setVariables( Map<String, String> variables );

/**
 * Set all ElementCatalog parameters into the view
 * @param elementCatalogParameters the ElementCatalog parameters
 */
void setElementCatalogParameters( Map<String, String> elementCatalogParameters);
}
```

The default BaseController implements WCSController to provide default WebCenter Sites functionalities to customers who have implemented Controllers that do not provide. WCSController makes the access to assets and other WebCenter Sites data easy. A new set of annotations allows customers to inject certain WebCenter Sites-specific properties into the Controller. Specifically, Oracle has added annotations to inject these properties: AssetReader, NavigationReader, ics variables, ics session variables, headers, query parameters, and ControllerInvoker (allows customers to pass a Controller name. It invokes the passed-in Controller in the current Controller). Sample Controllers that use annotations to provide basic common functionalites are as follows:

Hello World Controller

```
package oracle.webcenter.sites.controller

import COM.FutureTense.Common.ControllerException
import COM.FutureTense.Interfaces.Controller
import COM.FutureTense.Interfaces.DependenciesAwareModelAndView
import COM.FutureTense.Interfaces.IItem
import COM.FutureTense.Cache.AccessedItem
import COM.FutureTense.Interfaces.ModelAndViewInstance
import com.fatwire.assetapi.data.*
import com.fatwire.assetapi.site.NavigationReader
import com.openmarket.xcelerate.asset.*
import com.fatwire.assetapi.fragment.*

public class FSIIHelloWorld implements Controller
{
  private Map models = new HashMap();
  public Map getModels()
  {
    return models;
  }

  public DependenciesAwareModelAndView handleRequest() throws ControllerException
  {
    models.put("text", "Hello World");

    return new ModelAndViewInstance( models );
  }
}
```

This Controller implements the Controller interface and only sets a "Hello World" message into the Model as a simple demonstration.

AssetReader Controller

```
package oracle.webcenter.sites.controller

import COM.FutureTense.Common.ControllerException
import COM.FutureTense.Interfaces.Controller
import COM.FutureTense.Interfaces.DependenciesAwareModelAndView
import COM.FutureTense.Interfaces.ModelAndViewInstance
import com.fatwire.assetapi.data.*
import com.fatwire.assetapi.site.NavigationReader
import com.openmarket.xcelerate.asset.*
import com.fatwire.assetapi.fragment.*

public class FSIIAssetReader implements Controller
{
  @InitAssetReader (assetType="AVIArticle", assetId=1328196047241 )
  private AssetReader assetReader;

  @InitAssetReader ( select="name,description" )
  private AssetReader assetReaderWithoutId;

  @InitRequestValuesMap (type=RequestValuesType.VARIABLES)
  private Map<String, Object> icsVariables;

  private Map models = new HashMap();
  public Map getModels()
  {
    return models;
  }

  public DependenciesAwareModelAndView handleRequest() throws ControllerException
  {
    models.put("assetReader", assetReader.read());

models.put("assetReaderWithoutId",assetReaderWithoutId.forAsset( (String)icsVariables
.get("c"), Long.parseLong((String)icsVariables.get("cid"))).read());
    return new ModelAndViewInstance(models);

  }
}
```

AssetReader Controller uses three annotations to initialize two AssetReaders to enable them for use in the Controller. The first AssetReader is initialized with assetType and assetId passed in through annotation parameters. The second one initializes the AssetReader assetType/assetId inside the handleRequest method using the values from icsVariables. Each AssetReader reads the specified asset and puts the results into the Model for display.

NavigationReader Controller

```
package oracle.webcenter.sites.controller

import COM.FutureTense.Common.ControllerException
import COM.FutureTense.Interfaces.Controller
import COM.FutureTense.Interfaces.DependenciesAwareModelAndView
import COM.FutureTense.Interfaces.ModelAndViewInstance
import com.fatwire.assetapi.data.*
import com.fatwire.assetapi.site.NavigationReader
import com.openmarket.xcelerate.asset.*
```

```
import com.fatwire.assetapi.fragment.*

public class FSIINavigationReader implements Controller
{
  @InitNavigationReader (assetType = "Page", assetId = 1346043544347)
  private NavigationReader navigationReader;

  private Map models = new HashMap();
  public Map getModels()
  {
    return models;
  }

  public DependenciesAwareModelAndView handleRequest() throws ControllerException
  {
    models.put("navigationReader", navigationReader.read());
     return new ModelAndViewInstance(models);
 }
}
```

Like the AssetReader Controller, the NavigationReader Controller uses annotations to
initialize a NavigationReader in the Controller so it can be used by users to access the
Navigation information from WebCenter Sites.

Variables, Headers, Session Variables and Request Parameters

```
package oracle.webcenter.sites.controller

import COM.FutureTense.Common.ControllerException
import COM.FutureTense.Interfaces.Controller
import COM.FutureTense.Interfaces.DependenciesAwareModelAndView
import COM.FutureTense.Interfaces.ModelAndViewInstance
import com.fatwire.assetapi.data.*
import com.fatwire.assetapi.site.NavigationReader
import com.openmarket.xcelerate.asset.*
import com.fatwire.assetapi.fragment.*

public class FSIIVariablesAnnotations implements Controller
{
  @InitRequestValuesMap (type=RequestValuesType.HEADERS)
  private Map<String, Object> headers;

  @InitRequestValuesMap (type=RequestValuesType.VARIABLES)
  private Map<String, Object> icsVariables;

  @InitRequestValuesMap (type=RequestValuesType.QUERY_PARAMS)
  private Map<String, Object> params;

  @InitRequestValuesMap (type=RequestValuesType.SESSION_VARIABLES)
  private Map<String, Object> icsSessionVariables;

  private Map models = new HashMap();
  public Map getModels()
  {
    return models;
  }

  public DependenciesAwareModelAndView handleRequest() throws ControllerException
  {
    models.put("headers", headers);
    models.put("variables", icsVariables);
    models.put("sessionVariables",  icsSessionVariables);
```

```
      models.put("queryParams", params);
      return new ModelAndViewInstance(models);
  }
}
```

These annotations populate a set of variables in the Controller from headers, ics variables, ics session variables and request parameters to be used in the Controller.

Logging a Dependency from Controller

```
package oracle.webcenter.sites.controller

import COM.FutureTense.Common.ControllerException
import COM.FutureTense.Interfaces.Controller
import COM.FutureTense.Interfaces.DependenciesAwareModelAndView
import COM.FutureTense.Interfaces.IItem
import COM.FutureTense.Cache.AccessedItem
import COM.FutureTense.Interfaces.ModelAndViewInstance
import com.fatwire.assetapi.data.*
import com.fatwire.assetapi.site.NavigationReader
import com.openmarket.xcelerate.asset.*
import com.fatwire.assetapi.fragment.*

public class FSIILogDeps implements Controller
{
  private Map models = new HashMap();
  public Map getModels()
  {
    return models;
  }

  public DependenciesAwareModelAndView handleRequest() throws ControllerException
  {
    models.put("text", "Log Controller Deps");

    List<IItem> items = new ArrayList<IItem>();
    items.add( new AccessedItem("Deps logged from Controller"));
    return new ModelAndViewInstance(models, items);
  }
}
```

This Controller demonstrates how a user-defined dependency could be logged from Controller.

Displaying a View Markup from Controller

```
package oracle.webcenter.sites.controller

import COM.FutureTense.Common.ControllerException
import COM.FutureTense.Interfaces.Controller
import COM.FutureTense.Interfaces.DependenciesAwareModelAndView
import COM.FutureTense.Interfaces.ModelAndViewInstance
import com.fatwire.assetapi.data.*
import com.fatwire.assetapi.site.NavigationReader
import com.openmarket.xcelerate.asset.*
import com.fatwire.assetapi.fragment.*

public class FSIIViewMarkup implements Controller
{
  private Map models = new HashMap();
  public Map getModels()
  {
```

```
      return models;
    }

  public DependenciesAwareModelAndView handleRequest() throws ControllerException
  {
    models.put("text", "Some Text");
    return new ModelAndViewInstance(models, null, "Display this view markup text
from Controller", null);
  }
}
```

This Controller informs Webcenter Sites that original page is not displayed, instead, the text in the viewMarkupText returned from Controller is returned.

Redirecting the View to Another WebCenter Sites Page

```
package oracle.webcenter.sites.controller

import COM.FutureTense.Common.ControllerException
import COM.FutureTense.Interfaces.Controller
import COM.FutureTense.Interfaces.DependenciesAwareModelAndView
import COM.FutureTense.Interfaces.ModelAndViewInstance
import com.fatwire.assetapi.data.*
import com.fatwire.assetapi.site.NavigationReader
import com.openmarket.xcelerate.asset.*
import com.fatwire.assetapi.fragment.*

public class FSIIRedirectView implements Controller
{
  private Map models = new HashMap();
  public Map getModels()
  {
    return models;
  }

  public DependenciesAwareModelAndView handleRequest() throws ControllerException
  {
    return new ModelAndViewInstance(models, null, null, "FirstSiteII/
FSIIRedirectedView");
  }
}
```

This Controller instructs WebCenter Sites to redirect a page to another WebCenter Sites page content.

Calling Another Controller

```
package oracle.webcenter.sites.controller

import COM.FutureTense.Common.ControllerException
import COM.FutureTense.Interfaces.Controller
import COM.FutureTense.Interfaces.DependenciesAwareModelAndView
import COM.FutureTense.Interfaces.IItem
import COM.FutureTense.Cache.AccessedItem
import COM.FutureTense.Interfaces.ModelAndView
import COM.FutureTense.Interfaces.ModelAndViewInstance
import com.fatwire.assetapi.data.*
import com.fatwire.assetapi.site.NavigationReader
import com.openmarket.xcelerate.asset.*
import com.fatwire.assetapi.fragment.*
```

```
public class FSIICallAnotherController implements Controller
{
  @InjectControllerInvoker (controllerName = "FSIIHelloWorld", pagename =
"FirstSiteII/FSIIHelloWorld")
  private ControllerInvoker controllerInvoker;

  private Map models = new HashMap();
  public Map getModels()
  {
    return models;
  }

  public DependenciesAwareModelAndView handleRequest() throws ControllerException
  {
    ModelAndView modelAndView = controllerInvoker.invoke(new HashMap());

    models.putAll(modelAndView.getModel());

    return new ModelAndViewInstance( models);
  }
}
```

This Controller uses annotations to initialize a ControllerInvoker with the Controller name so that the passed-in Controller could be invoked inside this Controller.

14.3 Creating a Controller

Creating a Controller with the Create New Controller wizard in the Admin interface of Oracle WebCenter Sites is quick and easy. All you need to do is give your Controller a name and add its business logic to the wizard.

In a similar way, you can create Controllers with Oracle Developer Tools in an Eclipse IDE. See Managing WebCenter Sites Resources in Eclipse.

To create a Controller through the Admin interface:

1. On the New page of the WebCenter Sites Admin interface, click **New Controller** in the asset type table.

   The Create New Controller wizard opens.

2. In the **Name** field, enter a name for the new `Controller`.

3. (Optional) In the **Description** field, enter a description of the new `Controller`.

4. In the **Controller Logic** field, add your business logic code.

5. Click the **Save** icon.

For more information about developing `Controllers`, go to the Developer's Samples Website and choose **Controller** from the **Site Rendering** menu.

14.4 Creating a Template

The WebCenter Sites Admin interface's easy-to-use template form lets you can create a `Template` asset quickly.

To create a `Template` using the Admin interface:

1. Click **New**.

2. In the table on the right, click the **New Template** link next to **Template**.

3. Fill in the `Template` form.

4. Save the `Template`.

See Creating Template Assets.

14.5 Setting Up the Home Page

Design a modular home page for your website that uses common elements. You can use the common code of those elements in several locations or contexts.

For information about setting up a home page for your website, see these topics:

• Creating Basic Modular Design

• Home Element

14.6 Adding Site Navigation

Oracle WebCenter Sites provides Navigation Reader, Link Builder, and Bob Link Builder Java APIs that you can use to create a site navigation for your website.

You can add site navigation for your website with the following Java APIs:

• Navigation Reader

• Link Builder

• Blob Link Builder

For more information about these Java APIs, go to the Rendering API menu on the Developer's Samples Website.

15

Developing a Client-Side Website

Would you like to develop a website for highly interactive applications and mobile sites? Your site can be a responsive application, a single-page application, and much more. Oracle WebCenter Sites APIs let you create such websites to render on client side and run in the browser client.

Before proceeding further, it is recommended that you familiarize yourself with the WebCenter Sites Aggregate REST services. See the Aggregate REST API Reference for Oracle WebCenter Sites.

Topics:

- About Client-Side Websites
- REST Calls for Developing REST-Avisports: Examples

15.1 About Client-Side Websites

REST APIs enable you to build websites that render on the client side and execute entirely in the browser client. Such websites include, but are not limited to, single-page applications. These websites are particularly useful for highly interactive applications (or portions of websites) and mobile sites.

In client-side rendering, usually client-side libraries generate the mark up, which helps in limiting the interactions between the client and server to just data elements. This interaction takes place usually in the form of JSON. As users navigate from page to page, requests are made to the server for specific data components while the JavaScript to generate the markup stays loaded in the browser. REST APIs mirror features of server-side Java APIs in general.

REST services in WebCenter Sites are exposed to clients so that they can get data and build their custom sites using their Client side frameworks. Client-Side APIs enable you to build JavaScript-based web applications that have no server-side logic or JSPs. These APIs can aggregate asset data for an entire page in a single REST call. A single call is sufficient to retrieve content up to any level of depth, its related information, and even an entire website.

Responses to REST services can potentially be cached in the browser cache, which can lead to a better performing experience. These APIs let consumers retrieve data from sites and present it in their own templates, without having to use WebCenter Sites templates. This feature allows customers to use WebCenter Sites as a data storage.

15.2 REST Calls for Developing REST-Avisports: Examples

REST-avisports is a sample website that demonstrates client-side website development using the WebCenter Sites REST API. This sample website is included in the WebCenter Sites installation download.

To configure this website, see Configuring and Deploying the REST-avisports Sample Site in *Oracle Fusion Middleware Installing and Configuring Oracle WebCenter Sites*.

This section provides some examples of REST calls used in developing the REST-avisports client-side website, which is similar to the avisports sample site. These examples give site developers some ideas about efficiently developing a full-fledged client-side website using REST calls. Developing the REST-avisports website involves the following:

- Getting Navigation Menus

- Getting the Home Page

- Getting an Additional Website Page

- Calling an Article from a Page

- Calling a Collection Resource with Pagination

- Calling a Search Resource

- Calling Page Segments

15.2.1 Getting Navigation Menus

The following call renders Home and Skiing navigation menus on the avisports sample website:

```
http://<host>:<port>/<context>/REST/resources/v1/aggregates/avisports/navigation/
Default?
assetDepth=0&fields=Page(children,name,id);SiteNavigation(children)&expand=Page
```

This call uses the following query parameters: `assetDepth`, `fields`, and `expand`.

The response looks something like this:

Figure 15-1   Navigation Response

```
{
    "start" : "SiteNavigation:1052581735",
    "links" : [...],
    "SiteNavigation:1052581735" : {
        "children" : [
            {
                "start" : "Page:1327351719456",
                "Page:1327351719456" : {
                    "name" : "Home",
                    "id" : 1327351719456,
                    "children" : [
                        {
                            "Page:1329851332601" : {
                                "name" : "Surfing",
                                "id" : 1329851332601,
                                "parents" : [
                                ]
                            },
                            "start" : "Page:1329851332601"
                        },
                        {
                            "Page:1329326970440" : {
                                "name" : "Skiing",
                                "id" : 1329326970440,
                                "parents" : [
                                ]
                            },
                            "start" : "Page:1329326970440"
                        },
                        {...},
                        {...},
                        {...}
                    ],
                    "parents" : [
                    ]
                }
            }
        ],
        "parents" : [...]
    }
}
```

The navigation shown in this figure is rendered:

Figure 15-2   Navigation Rendered from the JSON Response

15.2.2 Getting the Home Page

The following call renders the Home page asset on the avisports sample website:

```
http://<host>:<port>/<context>/REST/resources/v1/aggregates/avisports/Page/
1327351719456?
assetDepth=1&fields=Page(banner,teaserImages,teaserText,bannerText);AVIImage(imageFil
e,width,height)&expand=Page,AVIImage
```

This call uses the following query parameters: `assetDepth`, `fields`, and `expand`.

The response looks something like this:

Figure 15-3   JSON Response to Render the Home Page

The page shown in this image is rendered:

Figure 15-4   Avisports Home Page Rendered from the JSON Response
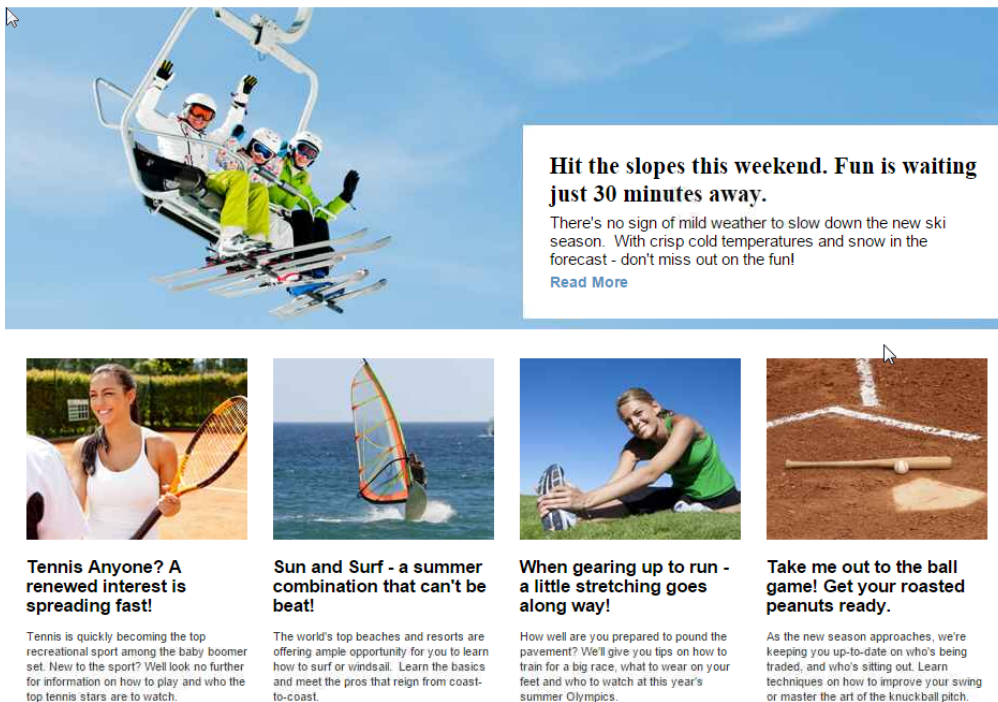
15.2.3 Getting an Additional Website Page

The following call renders the Skiing page on the avisports sample website:

```
http://<host>:<port>/<context>/REST/resources/v1/aggregates/avisports/Page/
1329326970440?
assetDepth=2&fields=AVIArticle(id,relatedImage,abstract,headline);Page(banner,titleCo
ntent1,titleContent2,Assoc_Named_contentList1,Assoc_Named_contentList2);AVIImage(imag
eFile,smallThumbnail,ClargeThumbnail);YouTube(externalid)&expand=Page,AVIImage,AVIArt
icle,YouTube
```

This call uses the following query parameters: `assetDepth`, `fields`, and `expand`.

The response looks something like this:

Figure 15-5   JSON Response to Render the Skiing Page

```
{
    "Page:1329326970440" : {
        "titleContent1" : "Top Stories In Skiing",
        "titleContent2" : "Also In The News",
        "banner" : {...},
        "Assoc_Named_contentList2" : [
            {
                "id" : 1328196047241,
                "type" : "AVIArticle"
            },
            {...}
        ],
        "Assoc_Named_contentList1" : [
            {
                "id" : 1366027008563,
                "type" : "YouTube"
            },
            {...}
        ],
        "parents" : [...]
    },
    "AVIImage:1327351719292" : {...},
    "AVIArticle:1330880479095" : {...},
    "AVIArticle:1328196048187" : {...},
    "YouTube:1366027008563" : {
        "externalid" : "K4DKyPfzQEU",
        "parents" : [
        ]
    },
    "AVIArticle:1328196047241" : {...},
    "AVIImage:1327351719187" : {...},
    "start" : "Page:1329326970440",
    "AVIImage:1328196052407" : {...},
    "AVIImage:1328196052470" : {...},
    "links" : [...]
}
```

The Skiing page shown in this image is rendered on the avisports sample website.

Figure 15-6   Skiing Page Rendered from the JSON Response
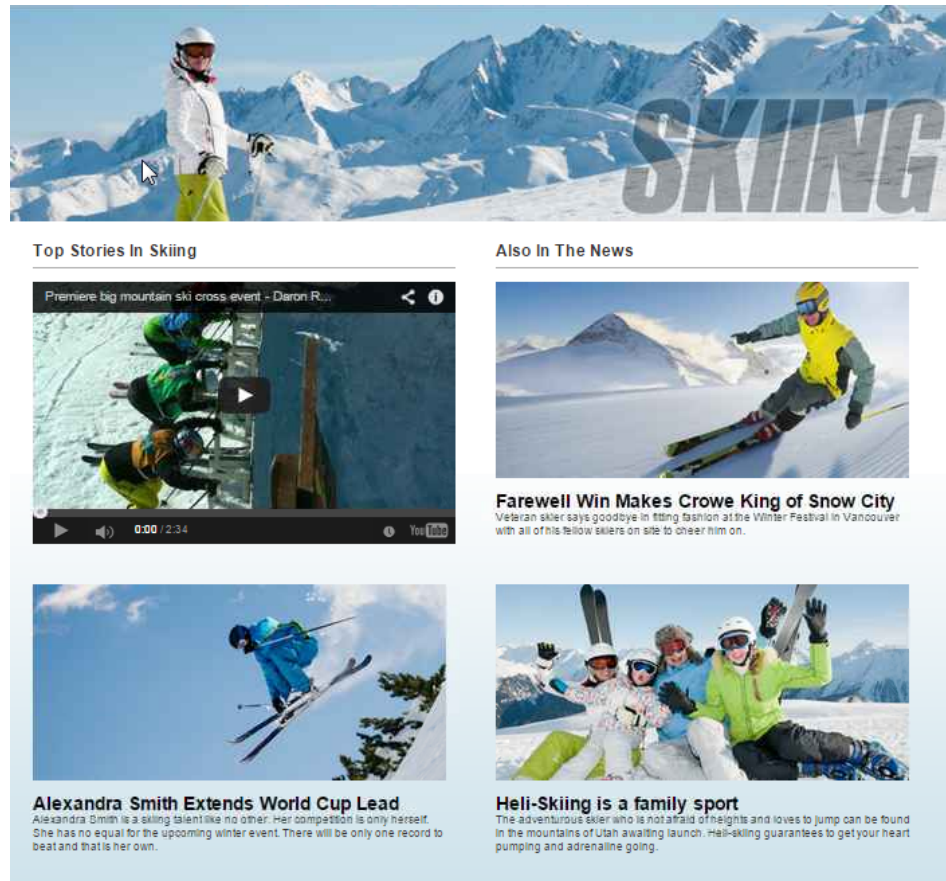
15.2.4 Calling an Article from a Page

The following call renders an article that is displayed in the **Also In the News** section on the Skiing page on the avisports sample website:

```
http://<host>:<port>/<context>/REST/resources/v1/aggregates/avisports/AVIArticle/
1328196047241?
assetDepth=2&fields=category,author,postDate,subheadline,relatedLinks,relatedStories,
relatedImage,body;AVIArticle(id,headline,abstract,relatedImage);AVIImage(imageFile,ca
ption,width,height,sidebarThumbnail)&expand=AVIArticle,AVIImage
```

This call uses the following query parameters: `assetDepth`, `fields`, and `expand`.

The response looks something like this:

Figure 15-7 *JSON Response to Render an Article Displayed on the Skiing Page*

The article shown in this figure is rendered:

Figure 15-8 *Article Rendered from the JSON Response*

15.2.5 Calling a Collection Resource with Pagination

The following call renders a collection resource with pagination (with navigation buttons on the bottom of the page) on the avisports sample website:

```
http://<host>:<port>/<context>/REST/resources/v1/aggregates/avisports/ContentQuery/
1395380847207/items?
assetDepth=1&fields=AVIArticle(id,headline,abstract,relatedImage);AVIImage(imageFile,
smallThumbnail)&expand=AVIArticle,AVIImage&offset=0&limit=8
```

This call uses the following query parameters: assetDepth, fields, and expand.

The response looks something like this:

Figure 15-9   JSON Response to Render a Collection Resource with Pagination

```
{
    "offset" : 0,
    "limit" : 50,
    "count" : 8,
    "hasMore" : true,
    "links" : [...],
    "items" : [
        {
            "AVIImage:1327351719292" : {...},
            "AVIArticle:1328196047241" : {
                "relatedImage" : {
                    "id" : 1327351719292,
                    "type" : "AVIImage"
                },
                "id" : 1328196047241,
                "abstract" : "Veteran skier says goodbye in fitting fashion at the Winter Festival in Vancouver with all of his fellow skiers
                 on site to cheer him on.",
                "headline" : "Farewell Win Makes Crowe King of Snow City",
                "parents" : [
                    {
                        "href" : "http://<host>:<port>/<context>/REST/resources/v1/aggregates/avisports/ArticleCategory/1327351719208",
                        "rel" : "assetReference",
                        "templated" : false,
                        "mediaType" : "",
                        "method" : "GET",
                        "profile" : ""
                    }
                ]
            },
            "start" : "AVIArticle:1328196047241"
        },
        {...},
        {...},
        {...},
        {...},
        {...},
        {...},
        {...}
    ]
}
```

The page shown in this figure is rendered:

Figure 15-10   Collection Resource Rendered from the JSON Response

15.2.6 Calling a Search Resource

The following call renders the search results (which is a collection resource) for the criterion "How To" on the avisports sample website:

```
http://<host>:<port>/<context>/REST/resources/v1/search/sites/avisports/types/
AVIArticle/assets?field:name:startswith=How+to
```

This call uses the following query parameters: `field name starts with`.

The JSON response looks something like this:

Figure 15-11   JSON Response to Render Search Results

```
{
    "offset" : 0,
    "limit" : 13,
    "count" : 13,
    "hasMore" : false,
    "links" : [...],
    "items" : [
        {
            "name" : "How to Play Your Best",
            "link" : {...},
            "description" : "",
            "id" : "AVIArticle:1330544205026"
        },
        {
            "name" : "How to Buy the Right Running Shoes",
            "link" : {...},
            "description" : "",
            "id" : "AVIArticle:1330880434605"
        },
        {
            "name" : "How to Run Hills Properly",
            "link" : {...},
            "description" : "",
            "id" : "AVIArticle:1330880441898"
        },
        {...},
        {...},
        {...},
        {...},
        {...},
        {...},
        {...},
        {...},
        {...},
        {...}
    ]
}
```

The page shown in this figure is rendered:

Figure 15-12   Search Results from the JSON Response
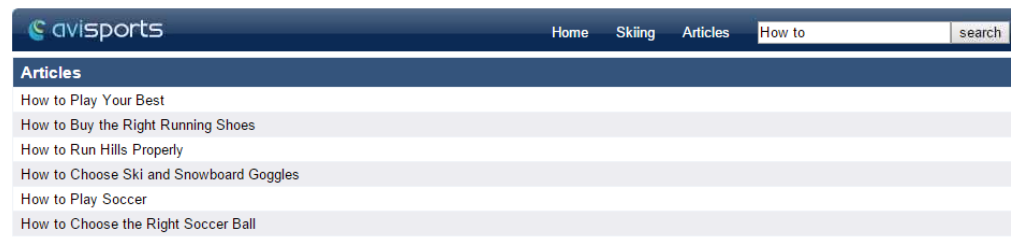
15.2.7 Calling Page Segments

Topics:

- Calling a Page Without Segments

- Calling a Page with Segments That Target Specific Visitors

- Calling a Page with Segments That Target Different Visitors

- Calling a Page with Segments That Target More Visitors

15.2.7.1 Calling a Page Without Segments

The following call renders the Running page without segments on the avisports sample website:

```
http://<host>:<port>/<context>/REST/resources/v1/aggregates/avisports/Page/
1361217259137?
assetDepth=1&fields=AVIArticle(id,relatedImage,abstract,headline);Page(banner,recomme
ndation);AVIImage(imageFile,smallThumbnail,largeThumbnail);AdvCols(items)&expand=Page
,AVIImage,AdvCols,AVIArticle
```

This call uses the following query parameters: `assetDepth`, `fields`, and `expand`.

The JSON response looks like this:

Figure 15-13   JSON Response to Render a Page Without Segments

The page shown in this image is rendered:

Figure 15-14   Page with No Segments Rendered

15.2.7.2 Calling a Page with Segments That Target Specific Visitors

The following call renders the Running page with segments that target men on the avisports sample website:

```
http://<host>:<port>/<context>/REST/resources/v1/aggregates/avisports/Page/
1361217259137?
assetDepth=1&fields=AVIArticle(id,relatedImage,abstract,headline);Page(banner,recomme
ndation);AVIImage(imageFile,smallThumbnail,largeThumbnail);AdvCols(items)&expand=Page
,AVIImage,AdvCols,AVIArticle&segments=Male
```

This call uses the following query parameters: `assetDepth`, `fields`, `expand`, and `segments`.

The JSON response looks like this:

Figure 15-15   JSON Response to Render a Page with Segments That Target Men

```
        "mapstyle" : "C",
        "createdby" : "fwadmin",
        "subtype" : "",
        "externaldoctype" : "",
        "id" : 1361217257773,
        "types" : [
            "_ALL_"
        ],
        "Manualrecs" : [...],
        "fw_uid" : "bae184f6-685a-42d9-a0a2-00543fec7394",
        "Dimension" : "[]",
        "selrulesetmap" : [...],
        "enddate" : null,
        "filename" : "",
        "name" : "Running Recommendation",
        "updateddate" : {
            "value" : "2013-04-08T22:20:03.390Z",
            "timezone" : "India Standard Time",
            "description" : ""
        },
        "style" : "",
        "fwtags" : [
        ],
        "items" : [
            {
                "AVIArticle:1363104047253" : {...},
                "metadata" : {
                    "confidence" : 100,
                    "rating" : 50
                },
                "start" : "AVIArticle:1363104047253",
                "AVIImage:1363103066572" : {...}
            },
            {...},
            {...},
            {...},
            {...},
            {...},
            {...},
            {...}
        ],
        "status" : "ED",
        "parents" : [...]
    }
}
```

The page shown in this figure is rendered:

Figure 15-16 Rendered Page with Segments That Target Men

15.2.7.3 Calling a Page with Segments That Target Different Visitors

The following call renders the Running page with segments that target women on the avisports sample website:

```
http://<host>:<port>/<context>/REST/resources/v1/aggregates/avisports/Page/
1361217259137?
assetDepth=1&fields=AVIArticle(id,relatedImage,abstract,headline);Page(banner,recomme
ndation);AVIImage(imageFile,smallThumbnail,largeThumbnail);AdvCols(items)&expand=Pag,
AVIImage,AdvCols,AVIArticle&segments=Female
```

This call uses the following query parameters: `assetDepth`, `fields`, `expand`, and `segments`.

The JSON response looks like this:

Figure 15-17   JSON Response to Render a Page with Segments That Target Women

```
      "Dimension-parent" : [...],
      "path" : "",
      "mapstyle" : "C",
      "createdby" : "fwadmin",
      "subtype" : "",
      "externaldoctype" : "",
      "id" : 1361217257773,
      "types" : [...],
      "Manualrecs" : [...],
      "fw_uid" : "bae184f6-685a-42d9-a0a2-00543fec7394",
      "Dimension" : "[]",
      "selrulesetmap" : [...],
      "enddate" : null,
      "filename" : "",
      "name" : "Running Recommendation",
      "updateddate" : {
          "value" : "2013-04-08T22:20:03.390Z",
          "timezone" : "India Standard Time",
          "description" : ""
      },
      "style" : "",
      "fwtags" : [
      ],
      "items" : [
          {
              "AVIArticle:1330880441898" : {...},
              "AVIImage:1329801673811" : {...},
              "metadata" : {
                  "confidence" : 100,
                  "rating" : 50
              },
              "start" : "AVIArticle:1330880441898"
          },
          {...},
          {...},
          {...},
          {...},
          {...},
          {...},
          {...}
      ],
      "status" : "ED",
      "parents" : [...]
   }
}
```

The page shown in this image is rendered:

Figure 15-18   Rendered Page with Segments That Target Women

15.2.7.4 Calling a Page with Segments That Target More Visitors

The following call renders the Running page with segments that target both women and men on the avisports sample website:

```
http://<host>:<port>/<context>/REST/resources/v1/aggregates/avisports/Page/
1361217259137?
assetDepth=1&fields=AVIArticle(id,relatedImage,abstract,headline);Page(banner,recomme
ndation);AVIImage(imageFile,smallThumbnail,largeThumbnail);AdvCols(items)&expand=Page
,AVIImage,AdvCols,AVIArticle&segments=Male,Female
```

This call uses the following query parameters: `assetDepth`, `fields`, `expand`, and `segments`.

The JSON response looks like this:

Figure 15-19   JSON Response to Render a Page with Segments That Target Women and Men

```
     "Dimension-parent" : [...],
     "path" : "",
     "mapstyle" : "C",
     "createdby" : "fwadmin",
     "subtype" : "",
     "externaldoctype" : "",
     "id" : 1361217257773,
     "types" : [
         "_ALL_"
     ],
     "Manualrecs" : [...],
     "fw_uid" : "bae184f6-685a-42d9-a0a2-00543fec7394",
     "Dimension" : "[]",
     "selrulesetmap" : [...],
     "enddate" : null,
     "filename" : "",
     "name" : "Running Recommendation",
     "updateddate" : {...},
     "style" : "",
     "fwtags" : [
     ],
     "items" : [
         {
             "AVIArticle:1363104047253" : {...},
             "metadata" : {
                 "confidence" : 100,
                 "rating" : 50
             },
             "start" : "AVIArticle:1363104047253",
             "AVIImage:1363103066572" : {...}
         },
         {...},
         {...},
         {...},
         {...},
         {...},
         {...},
         {...},
         {...},
         {...}
     ],
     "status" : "ED",
     "parents" : [...]
    }
}
```

The page shown in this image is rendered:

Figure 15-20   Rendered Page with Segments That Target Women and Men

16

Website Development with Tag Technologies

For backwards compatibility, Oracle WebCenter Sites continues to provide legacy tag technologies for developing websites. You can code with WebCenter Sites tags and Java methods, and for templates you can use XML and JSP.

For information about the legacy WebCenter Sites tag technologies, see these topics:

- About Choosing a Coding Language

- About the Oracle WebCenter Sites Context

- Understanding WebCenter Sites JSP

- Understanding WebCenter Sites XML

- Understanding WebCenter Sites Tags

- About Variables Supported in WebCenter Sites

- Other WebCenter Sites Storage Constructs

- About Values for Special Characters

> **Note:**
>
> For information about the newer technologies, see Website Development with the MVC Framework and APIs.

16.1 About Choosing a Coding Language

Several scripting and markup languages, including HTML, XML, JSP, CSS, and JavaScript let you write elements and templates. But, WebCenter Sites only evaluates XML and JSP. Choose your coding or markup language based on what the element or template that you are creating does.

XML or JSP are suitable for elements that display content that may change, such as a newspaper article. This is because such elements use logic to retrieve their content from the WebCenter Sites database, and thus are managed using WebCenter Sites XML or JSP tags.

You typically use HTML and XML for page layout and JSP and Java for logic. WebCenter Sites also has a Java API, which you will use in conjunction with WebCenter Sites JSP tags if you choose JSP as your coding language.

This table lists the situations to which each language is best suited:

Table 16-1   Coding Language Usage

Code	When to Use
XML	The element contains mostly text, with few loops and conditionals.
JSP	• The element requires conditional operators, or relational operators other than = or !=. • The element uses many loops. Loops perform better in JSP than in XML. • The element contains calls to Java code.

Note that elements written in XML or JSP can call any type of element, but you cannot mix XML and JSP in the same element. For example, an element written in either XML or JSP can call another element written in HTML, XML, or JSP. However, an element written in HTML cannot call an element written in XML or JSP.

16.2 About the Oracle WebCenter Sites Context

You code an Oracle WebCenter Sites project within the Oracle WebCenter Sites context. The WebCenter Sites context provides access to the Java servlets that compose WebCenter Sites, and to the WebCenter Sites Java objects whose methods and tags allow you access to WebCenter Sites functionality.

You code in the WebCenter Sites context no matter what language you code your project in; WebCenter Sites XML and JSP tags provide an easy-to use interface to the WebCenter Sites Java objects, so that even web designers with little or no Java experience can create WebCenter Sites web pages.

16.2.1 The ICS Object

When you are coding for Oracle WebCenter Sites, you often access the methods and tags of the Interface to WebCenter Sites (ICS) object. The ICS object encapsulates some WebCenter Sites core functionality, allowing you to access servlets that control the WebCenter Sites tree (the TreeManager servlet) and the input of data into the database (the CatalogManager servlet).

You also use ICS methods and tags to perform tasks such as creating and displaying variables and using if/then statements to perform tasks based on specified conditions. For information about ICS object's methods and tags, see the *Oracle Fusion Middleware Tag Reference for Oracle WebCenter Sites Reference*.

> **Note:**   The objects stored in the ICS object are vulnerable to the modifications made by different threads. Therefore, you must ensure that the ICS object is used in a thread safe manner.

16.2.2 The FTCS tag

Each WebCenter Sites element or template begins and ends with the ftcs tag. This tag creates the WebCenter Sites context, alerting WebCenter Sites that code contained within the opening and closing ftcs tags will contain WebCenter Sites tags and access ICS methods.

Elements and templates that you create using the WebCenter Sites user interface or the Oracle WebCenter Sites Explorer tool include the opening and closing ftcs tags after

the standard directives. You must code within the opening and closing `ftcs` tags; WebCenter Sites is unaware of any code which falls outside of these tags.

If you create element and template code using some other method, you must add the opening `ftcs` tag after your directives, and use the closing `ftcs` tag as the last line of your code.

16.3 Understanding WebCenter Sites JSP

JSP programmers use a set of standard tools, including directives, actions, and JSP objects that you too have access to. Sometimes, however, you must substitute a WebCenter Sites tag for a JSP directive or action, or access a WebCenter Sites object rather than one of JSP's implicit objects.

These topics detail the differences between standard JSP and WebCenter Sites JSP, and how standard JSP functionality maps to WebCenter Sites tags and methods:

- About the WebCenter Sites Standard Beginning
- About JSP Implicit Objects
- About JSP Syntax
- About JSP Actions
- About JSP Declarations
- About Scriptlets and Expressions
- About JSP Directives
- About Oracle WebCenter Sites Tag Libraries

16.3.1 About the WebCenter Sites Standard Beginning

Template assets, CSElement assets, and non-asset elements that you create using the WebCenter Sites user interface or Explorer are automatically seeded with a standard beginning.

The standard beginning for a JSP element in Explorer follows:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld" %>
<%@ taglib prefix="ics" uri="futuretense_cs/ics.tld" %>
<%@ taglib prefix="satellite" uri="futuretense_cs/satellite.tld" %>
<%//
// elementName
//
// INPUT
//
// OUTPUT
//%>
<%@ page import="COM.FutureTense.Interfaces.FTValList" %>
<%@ page import="COM.FutureTense.Interfaces.ICS" %>
<%@ page import="COM.FutureTense.Interfaces.IList" %>
<%@ page import="COM.FutureTense.Interfaces.Utilities" %>
<%@ page import="COM.FutureTense.Util.ftErrors" %>
<%@ page import="COM.FutureTense.Util.ftMessage"%>
<cs:ftcs>

<!-- user code here -->
```

```
</cs:ftcs>
```

Template and CSElement assets that you create using the WebCenter Sites user interface include a standard beginning similar to the preceding code sample. The standard beginning for these assets imports additional tag libraries for use with basic assets and includes tags that log dependencies between the Template and CSElement assets and the content that they render.

If you use a tool other than Explorer or the WebCenter Sites user interface to create your elements and templates, you must copy the standard beginning into your code verbatim.

The following sections explain the standard beginning for Explorer.

16.3.1.1 Taglib Directives

The following `taglib` directives import the base tag libraries that you will use with WebCenter Sites. Template and CSElement assets that you create using the WebCenter Sites user interface include additional taglib directives in seed code:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld" %>
<%@ taglib prefix="ics" uri="futuretense_cs/ics.tld" %>
<%@ taglib prefix="satellite" uri="futuretense_cs/satellite.tld" %>
```

The first directive imports the `ftcs1_0` tags, which create the FTCS context. These tags are used in each template or element that you create, and indicate that the code enclosed by them will be controlled by WebCenter Sites.

The second directive imports the `ics` tags, which provide access to the WebCenter Sites core functionality.

The third directive imports the satellite tags, which are for use with Satellite Server.

For more information about these tag libraries, see About Oracle WebCenter Sites Tag Libraries.

For information about commonly used tags that are found in these tag libraries, see Understanding WebCenter Sites Tags.

To add taglib directives to these defaults, modify and save the `OpenMarket/Xcelerate/AssetType/Template/ModelJsp.xml` file.

16.3.1.2 Page Directives

The following page directives import the base Java interfaces that you will use with WebCenter Sites:

```
<%@ page import="COM.FutureTense.Interfaces.FTValList" %>
<%@ page import="COM.FutureTense.Interfaces.ICS" %>
<%@ page import="COM.FutureTense.Interfaces.IList" %>
<%@ page import="COM.FutureTense.Interfaces.Utilities" %>
<%@ page import="COM.FutureTense.Util.ftErrors" %>
<%@ page import="COM.FutureTense.Util.ftMessage"%>
```

The first page directive imports the FTValList interface, which creates a list of name/value pairs that you use to pass arguments to WebCenter Sites subsystems like the CatalogManager and TreeManager.

The second page directive imports the `ICS` interface, which provides access to the core WebCenter Sites functionality.

The third page directive imports the IList interface, which contains the methods to access the rows in a WebCenter Sites query or list object. It also contains the methods that a third party must implement when attempting to construct and register a list object for use within an WebCenter Sites XML page.

The fourth page directive imports the Utilities interface, which provides a simple interface for some common tasks such as formatting dates, reading and writing files, and sending email.

The fifth page directive imports the ftErrors class, which contains error codes.

The sixth page directive imports the ftMessage class, which contains error messages used by WebCenter Sites.

To add page directives to the standard directives for JSP elements, modify and save the OpenMarket/Xcelerate/AssetType/Template/ModelJsp.xml file.

16.3.1.3 The cs:ftcs Tag

Each WebCenter Sites JSP template or element must have the cs:ftcs tag as its first and last tags. This tag creates the WebCenter Sites context, alerting WebCenter Sites that code contained within the opening and closing cs:ftcs tags will contain WebCenter Sites tags.

You must code within the opening and closing cs:ftcs tags; WebCenter Sites is unaware of any code which falls outside of these tags.

16.3.2 About JSP Implicit Objects

JSP provides several implicit objects that are available for developers to use. In the WebCenter Sites context, however, you are often dealing with the WebCenter Sites objects, and should use WebCenter Sites JSP tags and Java methods to access these objects, instead of using JSP's implicit objects.

This table maps JSP's implicit objects and some commonly used methods to the WebCenter Sites tag or method that you should use to replace them.

Table 16-2   JSP Implicit Object Mapping

Object	Method	WebCenter Sites Tag or Method
request	getParameter	ics:getvar tag
request	getParameterNames	ICS.GetVars() method
request	getCookie	ics:getCookie tag
response	addCookie	satellite:cookie tag
session	getAttribute	ics:getssvar tag
session	setAttribute	ics:setssvar tag
out	println	ics:getvar tag or render:stream tag

16.3.3 About JSP Syntax

Oracle WebCenter Sites uses standard JSP syntax. When you are nesting tags, for example, using a JSP expression as the value of a JSP tag's parameter, remember to use single quotes to contain the expression, as in the following example:

```
name='<%=ics.GetVar("myVariable")%>'
```

16.3.4 About JSP Actions

Standard JSP allows developers to use several different actions. This table describes what actions should be replaced with WebCenter Sites tags and which can be used as usual.

Table 16-3   JSP Actions vs. WebCenter Sites Tags

Action	WebCenter Sites
`<jsp:forward>`	Use the `render:satellitepage` or `render:callelement` tags instead.
`<jsp:getproperty>`	Use this for custom Java Beans. to find the value of one of the WebCenter Sites properties, use the `<ics:getproperty>` tag.
`<jsp:include>`	Use the `render:satellitepage` or `render:callelement` tags instead.
`<jsp:setProperty>`	Use this to set properties in custom Java Beans. Use the WebCenter Sites Property Management Tool to set WebCenter Sites properties.
`<jsp:useBean>`	Use this for custom Java Beans.

16.3.5 About JSP Declarations

In standard JSP, you usually declare variables within a JSP declaration. In WebCenter Sites, you use the `ics:setvar` tag to declare variables that are available in the WebCenter Sites context.

For more information about WebCenter Sites variables, see About Variables Supported in WebCenter Sites.

16.3.6 About Scriptlets and Expressions

You can use scriptlets and expressions without any variation from normal JSP usage.

When you use an expression as the value of the parameter for a WebCenter Sites JSP tag, however, be sure that you nest quotation marks correctly, as described in About JSP Syntax.

16.3.7 About JSP Directives

When you are coding JSP in a WebCenter Sites context, there are some caveats for using directives, which are outlined in this table:

Table 16-4   JSP Directives vs. WebCenter Sites Tags

Directive	WebCenter Sites
`IncludeDirective`	Use the `render:satellitepage` or `render:callelement` tags to include other files in your JSP pages.
`Page Directive`	Elements or templates that you create using the WebCenter Sites user interface or the Oracle WebCenter Sites Explorer tool are automatically seeded with standard page directives. In addition to the standard directives, you must add one other page directive to set the `contentType` for each WebCenter Sites element or template that you create. Set your page's content type to text/html and the character set to UTF-8 by providing the following page directive as the first line of every WebCenter Sites JSP file: `<%@ page contentType="text/html; charset=UTF-8" %>`
`Taglib Directive`	WebCenter Sites automatically seeds your templates and elements with commonly used taglib directives. You can add additional WebCenter Sites taglib directives to an element or Template asset as needed.

16.3.8 About Oracle WebCenter Sites Tag Libraries

WebCenter Sites has a series of JSP tag libraries that correspond to functions in the WebCenter Sites APIs.

The table below lists the WebCenter Sites tag libraries and describes their functions. Use this table as a reference when deciding which tag libraries to import into your JSPs.

Table 16-5   Tag Libraries for Both Basic and Flex Assets

Tag Library	Description
`acl.tld`	Tags for creating and manipulating Access Control Lists.
`date.tld`	Tags that convert dates with year, month, day, and optional hour, minute, and am/pm fields into epoch format long integers representing milliseconds since Jan 1, 1970, 0:00 GMT. Date tags also convert long integers into dates.
`dir.tld`	Directory Services tags.
`ftcs1_0.tld`	Tags that create the FTCS context. These tags are used in each template or element that you create, and indicate that the code enclosed by them will be controlled by WebCenter Sites.
`ics.tld`	Tags which provide access to core WebCenter Sites functionality, including access to the CatalogManager and TreeManager commands, and basic coding constructs like if/then statements.
`insite.tld`	Tags for Web Mode.

Table 16-5   (Cont.) Tag Libraries for Both Basic and Flex Assets

Tag Library	Description
localestring.tld	Tags for localizing text strings.
name.tld	Tags that access the name of the user who is currently logged in to WebCenter Sites and manipulate user names in directory services.
object.tld	Tags for manipulating WebCenter Sites objects.
property.tld	Tags for retrieving values from WebCenter Sites property files.
render.tld	Tags that render basic assets.
tags for working with satellite server	Many of these tags have RENDER equivalents (as defined in render.tld) that are preferred for building sites with WebCenter Sites.
soap.tld	WebCenter Sites SOAP tags.
time.tld	Tags that get and set the timing for determining the performance of elements.
user.tld	Tags to log users in and out of WebCenter Sites.
webservices.tld	Web services tags that allow you to consume certain types of public websites as part of a WebCenter Sites page.

This table shows data on tag libraries for basic assets.

Table 16-6   Tag Libraries for Basic Assets

Tag Library	Description
asset.tld	Tags that retrieve and manipulate basic assets.
siteplan.tld	Tags that allow access to the site navigation tree. You use these tags to create navigation for a site that uses basic assets.

This table shows data on tag libraries for flex assets.

Table 16-7   Tag Libraries for Flex Assets

Tag Library	Description
assetset.tld	Tags for creating assetsets with flex assets.
blobservice.tld	Tags for retrieving and manipulating blobs that are attributes of flex assets.
calculator.tld	Tags that provide basic calculator and Boolean functions.

Table 16-7   (Cont.) Tag Libraries for Flex Assets

Tag Library	Description
`cart.tld`	Tags that allow you to add, delete, and otherwise manipulate items in a shopping cart object.
`cartset.tld`	Tags that allow you store, retrieve, delete, and list shopping cart objects for a registered buyer.
`commercecontext.tld`	Tags that access the objects in the visitor context.
`currency.tld`	Tags that convert floating point values and currency strings, and perform formatting and rounding operations on currency strings.
`decimal.tld`	Tags that format floating point values as decimal objects in different locales.
`hash.tld`	Tags that allow you to cast an IList as a hash table and search it by key.
`listobject.tld`	Tags that construct WebCenter Sites resultset lists, which are used throughout your elements as arguments for other tags.
`locale1.tld`	Tags that generate a locale object, which is used to describe the locale for various other tags in the system.
`misc.tld`	Miscellaneous tags, including a tag that returns the names of all the columns in an input list
`searchstate.tld`	Tags for creating searchstates to constrain groups of flex assets (assetsets).
`session.tld`	A tag that flushes all stored objects for a given session.
`string.tld`	Tags that perform string manipulations.
`textformat.tld`	Tags that format text.
`vdm.tld`	Visitor Data Management tags, which enable you to record and retrieve information about website visitors from WebCenter Sites, or from other databases.

16.3.8.1 Tag Libraries for Both Basic and Flex Assets

Table 16-8   Tag Libraries for Both Basic and Flex Assets

Tag Library	Description
`acl.tld`	Tags for creating and manipulating Access Control Lists.
`date.tld`	Tags that convert dates with year, month, day, and optional hour, minute, and am/pm fields into epoch format long integers representing milliseconds since Jan 1, 1970, 0:00 GMT. Date tags also convert long integers into dates.

Table 16-8   (Cont.) Tag Libraries for Both Basic and Flex Assets

Tag Library	Description
`dir.tld`	Directory Services tags.
`ftcs1_0.tld`	Tags that create the FTCS context. These tags are used in each template or element that you create, and indicate that the code enclosed by them will be controlled by WebCenter Sites.
`ics.tld`	Tags which provide access to core WebCenter Sites functionality, including access to the CatalogManager and TreeManager commands, and basic coding constructs like if/then statements.
`insite.tld`	Tags for Web Mode.
`localestring.tld`	Tags for localizing text strings.
`name.tld`	Tags that access the name of the user who is currently logged in to WebCenter Sites and manipulate user names in directory services.
`object.tld`	Tags for manipulating WebCenter Sites objects.
`property.tld`	Tags for retrieving values from WebCenter Sites property files.
`render.tld`	Tags that render basic assets.
`tags for working with satellite server`	Many of these tags have RENDER equivalents (as defined in `render.tld`) that are preferred for building sites with WebCenter Sites.
`soap.tld`	WebCenter Sites SOAP tags.
`time.tld`	Tags that get and set the timing for determining the performance of elements.
`user.tld`	Tags to log users in and out of WebCenter Sites.
`webservices.tld`	Web services tags that allow you to consume certain types of public websites as part of a WebCenter Sites page.

16.3.8.2 Tag Libraries for Basic Assets

Table 16-9   Tag Libraries for Basic Assets

Tag Library	Description
`asset.tld`	Tags that retrieve and manipulate basic assets.
`siteplan.tld`	Tags that allow access to the site navigation tree. You use these tags to create navigation for a site that uses basic assets.

16.3.8.3 Tag Libraries for Flex Assets

Table 16-10   Tag Libraries for Flex Assets

Tag Library	Description
assetset.tld	Tags for creating assetsets with flex assets.
blobservice.tld	Tags for retrieving and manipulating blobs that are attributes of flex assets.
calculator.tld	Tags that provide basic calculator and Boolean functions.
cart.tld	Tags that allow you to add, delete, and otherwise manipulate items in a shopping cart object.
cartset.tld	Tags that allow you store, retrieve, delete, and list shopping cart objects for a registered buyer.
commercecontext.tld	Tags that access the objects in the visitor context.
currency.tld	Tags that convert floating point values and currency strings, and perform formatting and rounding operations on currency strings.
decimal.tld	Tags that format floating point values as decimal objects in different locales.
hash.tld	Tags that allow you to cast an IList as a hash table and search it by key.
listobject.tld	Tags that construct WebCenter Sites resultset lists, which are used throughout your elements as arguments for other tags.
locale1.tld	Tags that generate a locale object, which is used to describe the locale for various other tags in the system.
misc.tld	Miscellaneous tags, including a tag that returns the names of all the columns in an input list
searchstate.tld	Tags for creating searchstates to constrain groups of flex assets (assetsets).
session.tld	A tag that flushes all stored objects for a given session.
string.tld	Tags that perform string manipulations.
textformat.tld	Tags that format text.
vdm.tld	Visitor Data Management tags, which enable you to record and retrieve information about website visitors from WebCenter Sites, or from other databases.

For complete descriptions of the WebCenter Sites tags used for template development, see the *Oracle Fusion Middleware Tag Reference for Oracle WebCenter Sites Reference*.

16.4 Understanding WebCenter Sites XML

WebCenter Sites XML uses the standard XML syntax and is defined by the `futuretense_cs.dtd`. As with WebCenter Sites JSP tags, WebCenter Sites XML tags provide access to WebCenter Sites servlets and objects.

See these topics for what you need to be aware of when coding with WebCenter Sites XML:

16.4.1 WebCenter Sites Standard Beginning

Templates and elements that you create using the Admin interface or the Oracle WebCenter Sites Explorer tool are seeded with a standard beginning.

```
<?xml version="1.0" ?>
<!DOCTYPE ftcs SYSTEM "futuretense_cs.dtd">
<ftcs version="1.2">
</ftcs>
```

If you use a tool other than the WebCenter Sites user interface or the Explorer tool to create your elements and templates, you must copy this code into them verbatim.

The following sections explain this standard beginning.

16.4.1.1 XML Version and Encoding

The first line in any WebCenter Sites XML template or element must set the XML version, as follows:

```
<?xml version="1.0"?>
```

Note that in order for your element to run, `<?xml version="1.0"?>` must be the first line in the element, with no spaces before the text. The line must also have a hard return at the end, placing it on its own line.

Set the encoding for this template or element as follows:

```
<?xml version="1.0" encoding="utf-8"?>
```

16.4.1.2 The DTD File

WebCenter Sites XML is defined by the `futuretense_cs.dtd` file. You must import this file into each WebCenter Sites element or template that you code by entering the following line immediately after the XML version statement:

```
<!DOCTYPE ftcs SYSTEM "futuretense_cs.dtd">
```

16.4.1.3 The FTCS Tag

Each WebCenter Sites XML template or element must have the `ftcs` tag as its first and last tags. This tag creates the WebCenter Sites context, alerting WebCenter Sites that code contained within the opening and closing `ftcs` tags will contain WebCenter Sites tags.

You must code within the opening and closing `ftcs` tags; WebCenter Sites is unaware of any code which falls outside of these tags.

16.4.2 XML Entities and Reserved Characters

Because symbols such as < and > are reserved characters in XML, you must not place them in your content. For example, the following code confuses the XML parser because the less-than operator (<) is used inside text:

```
<P>4 < 7</P>
```

You must use character entities in place of reserved characters. Character entities begin with &# and end with a semicolon. Between the &# and the semicolon, you specify the decimal Latin-1 (a superset of ASCII) value of the character. For example, the decimal Latin-1 value of the < character is 60, so the correct way to code the preceding line in XML is:

```
<P>4 &#60; 7</P>
```

See About Values for Special Characters for a list of these character entities.

> **Note:**
>
> White spaces can cause issues when using JSP to generate an XML response. It is not as simple as adding the XML as a content-type to the server to allow the response to work. The JSP must be configured in a way that white space will not be inserted into the beginning of the XML file.

16.4.3 XML Parsing Errors

The XML parser that processes WebCenter Sites tags ensures that the tags are syntactically correct. This simplifies tracking down hard-to-find problems related to tagging syntax errors. A misspelled tag name is not reported as an error. This is because the XML parser doesn't require all tag names to exist in the DTD.

When a page request is made to a WebCenter Sites system and an XML syntax error is detected, the results streamed back will contain useful information to help you locate the problem. The results include a general error description, followed by the line/column location of the error. For example, the following error reports a bad parameter name:

```
Illegal attribute name NAM Illegal attribute name NAM
Location: null(6,11)
Context:
```

And the next error reports an incorrect tag nesting:

```
Close tag IF does not match start tag THEN Close tag IF does not match start tag
THEN
Location: null(13,3)
Context:
```

The XML parser also detects run-time errors. These are errors where the XML tags are syntactically correct, however, some error in the structure is detected during processing. For example, the following error reports an invalid use of ARGUMENT:

```
Failed to run template:c:\FutureTense\elements\dan.xml Runtime error Argument
invalid [Argument 5]
Containing tag: FTCS
```

16.5 Understanding WebCenter Sites Tags

WebCenter Sites has an extensive set of tags in both JSP and XML that allow you to access the various functions of WebCenter Sites and its product family. You use these tags in conjunction with HTML, Java, JavaScript, and custom tags, to code your website.

This section provides an overview of the tags that you are most likely to use in your Template assets and elements. For information about WebCenter Sites tags, see the *Oracle Fusion Middleware Tag Reference for Oracle WebCenter Sites Reference*.

The tags discussed here are arranged by usage, as follows:

- Tags That Create the WebCenter Sites Context

- Tags That Handle Variables

- Tags That Call Pages and Elements

- Tags That Create URLs

- Tags That Control Caching

- Tags That Set Cookies

- Programming Construct Tags

- Tags That Manage Compositional and Approval Dependencies

- Tags That Retrieve Information About Basic Assets

- Tags That Create Assetsets (Flex Assets)

- Tags That Create Searchstates (Flex Assets)

16.5.1 Tags That Create the WebCenter Sites Context

The following tags create the WebCenter Sites context in which you code. You use these tags in every template or element that you write.

FTCS (XML)	ftcs1_0:ftcs (JSP)
`<FTCS>`	`<ftcs1_0:ftcs>`
`</FTCS>`	`</ftcs1_0:ftcs>`

The `ftcs` tag creates the WebCenter Sites context. The opening `ftcs` tag should be the first tag in your code, and the closing `ftcs` tag should be the last tag in your code. WebCenter Sites is unaware of anything that falls outside of the opening and closing `ftcs` tags. Consequently, content outside the tags is not cached, and the tags will not operate correctly.

16.5.2 Tags That Handle Variables

The following tags handle variables in WebCenter Sites.

CSVAR (XML)	ics:getvar (JSP)
`<CSVAR NAME="variableName"/>`	`<ics:getvar name="variableName"/>`

CSVAR displays the value of a variable, session variable, built-in, or counter.

SETVAR (XML)	ics:setvar (JSP)
`<SETVAR NAME="variableName" VALUE="variableValue"/>`	`<ics:setvar name="variableName" value="variableValue"/>`

SETVAR sets the value of a regular, WebCenter Sites variable. The value of the variable exists for the duration of the page evaluation unless it is explicitly deleted using REMOVEVAR.

SETSSVAR (XML)	ics:setvar (JSP)
`<SETSSVAR NAME="variableName" VALUE="variableValue"/>`	`<ics:setssvar name="variableName" value="variableValue"/>`

SETSSVAR sets a session variable.

REPLACEALL (XML)	ics:resolvevariables (JSP)
`<REPLACEALL NAME="variableName" VALUE="variableValue"/>`	`<ics:resolvevariables name="variableName" [output="variable name"] [delimited="true|false"]/>`

REPLACEALL and ics:resolvevariables resolve multiple WebCenter Sites variables. In other words, when you want to use WebCenter Sites variables in HTML tags, you use these tags to resolve the variables.

See WebCenter Sites, see About Variables Supported in WebCenter Sites.

16.5.3 Tags That Call Pages and Elements

Use the following tags to call elements or templates.

> **Note:**
>
> CACHECONTROL, used below, has been deprecated.

RENDER.SATELLITEPAGE (XML)	render:satellitepage (JSP)

`<RENDER.SATELLITEPAGE`
`PAGENAME="nameOfPageEntry"`
`[CACHECONTROL="expiration_date_an`
`d_time"]`
`[ARGS_var1="value1"]/>`
	`<render:satellitepage`
	`pagename="nameOfPageEntry"`
	`[cachecontrol="expiration_date_an`
	`d_time"]>`
	`<[render:argument`
	`name="variable1"`
	`value="value1"]/>`
	`</render:satellitepage>`

RENDER.SATELLITEPAGE requests a WebCenter Sites pagelet and caches that pagelet in both WebCenter Sites and Satellite Server, if the pagelet is not in cache. To call a page or pagelet without caching it individually, use the RENDER.CALLELEMENT tag. The RENDER.SATELLITEPAGE tag has a stacked scope, so the only variables available to the page are ones that you explicitly pass in.

RENDER.CALLELEMENT (XML)	render:callelement (JSP)

`<RENDER.CALLELEMENT`
`ELEMENTNAME="nameOfElement"`
`[ARGS_var1="value"]/>`
	`<ics:callelement element="element`
	`name">`
	`<ics:argument name="argument`
	`name" value="arg value"/>`
	`</ics:callelement>`

RENDER.CALLELEMENT is similar to the RENDER.SATELLITEPAGE tag in that both tags call other WebCenter Sites code, either in an element or in a page. However, code called by RENDER.CALLELEMENT does not get cached as an individual page or pagelet on Satellite Server.

Use RENDER.CALLELEMENT to process the content of an element that you wrote for the WebCenter Sites Content Applications and you want the scope of that element to be stacked. The element must exist in the ElementCatalog.

16.5.4 Tags That Create URLs

RENDER.GETPAGEURL (XML)	render:getpageurl (JSP)

`<RENDER.GETPAGEURL`
`OUTSTR="myURL"`
`PAGENAME="SiteCatalogPageEntry"`
`cid="IDofAsset"`
`[p="IDofParentPage"]`
`[c="AssetType"]`
`[ADDSESSION="true"]`
`[DYNAMIC="true"]`
`[PACKEDARGS="stringFromPACKARGSta`
`g"]`
`[ARGS_xxx="y"]/>`
	`<render:getpageurl`
	`outstr="myURL"`
	`pagename="SiteCatalogPageEntry"`
	`cid="IDofAsset"`
	`[p="IDofParentPage"]`
	`[c="AssetType"]`
	`[addsession="true"]`
	`[dynamic="true"]`
	`[packedargs="stringFromPACKARGSta`
	`g"]>`
	`<[render:argument name="xxx"`
	`value="yyy"]/>`
	`</render:getpageurl>`

This tag creates a URL for an asset, processing the arguments passed to it into a URL-encoded string and returning it as the variable specified by the OUTSTR parameter. If rendermode is set to export, the tag creates a file name for a static HTML file (unless you specify that you want a dynamic URL). If rendermode is set to live, the tag creates a dynamic URL.

RENDER.SATELLITEBLOB (XML)	render:satelliteblob (JSP)

`<RENDER.SATELLITEBLOB`
`SERVICE="HTMLtagName"`
`BLOBTABLE="blobTable"`
`BLOBKEY="primaryKeyName"`
`BLOBWHERE="primaryKeyValue"`
`BLOBCOL="columnName"`
`BLOBHEADERNAMEN="headername"`
`BLOBHEADERVALUEN="mimetype"`
`[ARGS_format1="5"]`
`[CACHECONTROL="expirationDateAndTime"]/>`
	`<render:satelliteblob`
	`service="HTMLtagName"`
	`blobtable="blobTable"`
	`blobkey="primaryKeyName"`
	`blobwhere="primaryKeyValue"`
	`blobcol="columnName"`
	`blobheadernameN="headername"`
	`blobheadervalueN="mimetype"`
	`[cachecontrol="expirationDateAndTime"]>`
	`<[render:argument name="format1" value="5"]/>`
	`</render:satelliteblob>`

This tag creates an HTML tag with a BlobServer URL for assets that are blobs. For example, imagefile assets are blobs stored in the WebCenter Sites database which means they must be served by the BlobServer servlet. This tag creates an HTML tag that instructs a browser how to find and format the specified blob.

16.5.5 Tags That Control Caching

The following tag lets you control whether the output of the current template or element gets cached.

ics.disablecache (XML)	ics:disablecache (JSP)
`<ics.disablecache/>`	`<ics:disablecache/>`

Use ics.disable cache in conjunction with if/then statements that check for error conditions; if an error is present, the resulting rendered page is not cached. For more information and code samples for the ics.disablecache tag, see Ensuring that Incorrect Pages Are Not Cached.

16.5.6 Tags That Set Cookies

The following tag sets cookies in WebCenter Sites.

satellite.cookie (XML)	satellite:cookie (JSP)
```<satellite.cookie name="cookie_name" value="cookie_value" timeout="timeout" secure="true|false" url="URL" [domain="domain"]/>```	```<satellite:cookie> <satellite:parameter name='name' value='cookie_name'/> <satellite:parameter name='value' value='cookie_value'/> <satellite:parameter name='timeout' value='cookie_timeout'/> <satellite:parameter name='secure' value='true| false'/> <satellite:parameter name='url' value='url'> </satellite:cookie>```

`satellite.cookie` sets a cookie on the user's browser. This tag is the only way to set cookies in either XML or JSP.

16.5.7 Programming Construct Tags

The following tags allow you to use basic programming constructs.

IF/THEN/ELSE (XML)	ics:if/ics:then/ics:else (JSP)
```<IF COND="LOGICAL_EXPRESSION"> <THEN> tags and/or text </THEN> <ELSE> tags and/or text </ELSE> </IF>```	```<ics:if condition="logical expression"> <ics:then> tags and/or text </ics:then> <ics:else> tags and/or text </ics:else> </ics:if>```

`IF`, `THEN`, `ELSE` determine conditions. You typically use these tags to determine the value of a variable.

LOOP (XML)	ics:listloop (JSP)
```<LOOP [FROM="START"] [COUNT="LOOP_TIMES"] [LIST="LIST_NAME"] [UNTIL="END"]> ... </LOOP>```	```<ics:listloop listname="some list" [maxrows="number of loops"] [startrow="start row"] [endrow="end row"]/>```

`LOOP` and `ics:listloop` iterate through items in a list. Remember that excess code within these tags affects the performance of the template. Whenever possible, keep statements that do not have to be repeated outside the `LOOP` tags.

16.5.8 Tags That Manage Compositional and Approval Dependencies

For more information about compositional and approval dependencies, see About Dependencies.

RENDER.LOGDEP (XML)	render:logdep (JSP)

`<RENDER.LOGDEP ASSET="asset name"`
`CID="asset id"`
`C="asset type"/>`
	`<render:logdep asset="asset name"`
	`cid="asset id"`
	`c="asset type"/>`

Use the `RENDER.LOGDEP` tag if your template uses tags that obtain an asset's data without loading the asset, such as `ASSET.CHILDREN`.

RENDER.UNKNOWNDEPS (XML)	render.unknowndeps (JSP)
`<RENDER.UNKNOWNDEPS/>`	`<render:unknowndeps/>`

Use the `RENDER.UNKNOWNDEPS` tag if a page has a query or some other indeterminate connection to its dependent assets. This tag causes the page or pagelet to be regenerated at every publish because the dependencies cannot be determined. This means that you should use this tag sparingly.

RENDER.FILTER (XML)	render:filter (JSP)

`<RENDER.FILTER LIST="list name"`
`LISTVARNAME="output list name"`
`LISTDICOL="assetID column"`
`[LISTTYPECOL="assettype column"]`
`[TYPE="asset type"]`
`[ID="asset id"]`
`[VARNAME="output variable"/>`
	`<render:filter list="list name"`
	`listvarname="output list name"`
	`listidcol="assetID column"`
	`[listtypecol="assettype column"]`
	`[type="asset type"]`
	`[id="asset id"]`
	`[varname="output variable"/>`

Use the `RENDER.FILTER` tag to check for unapproved assets and prevent them from being included in the exported page. This tag filters either a single asset or list of assets by comparing each asset ID against the `assetid` column in the `ApprovedAssets` database table. During export rendering, it filters what can be published based on approval status. During live rendering, `RENDER.FILTER` does nothing. Use this tag whenever you have a database query for a list of assets in your template.

16.5.9 Tags That Retrieve Information About Basic Assets

ASSET.LOAD (XML)	asset:load (JSP)

`<ASSET.LOAD`
`NAME="assetName"`
`TYPE="assetType"`
`OBJECTID="object.id"`
`[FIELD="fieldName"]`
`[VALUE="fieldValue"]`
`[DEPTYPE="EXACT, EXISTS,`
`or GREATER"]/>`
	`<asset:load`
	`name="assetName"`
	`type="assetType"`
	`objectid="object.id"`
	`[field="fieldName"]`
	`[value="fieldValue"]`
	`[deptype="exact,exists,or`
	`greater"]/>`

This tag queries the database for a specific asset and then loads the asset's data into memory as an object. The object is then available to your elements until either the session is flushed or the name that is assigned to the object is overwritten.

The scope of the object names that you assign to loaded assets is **global.** Be sure to use unique object names so that your elements do not overwrite objects by mistake. A convenient naming convention is to include the element name in the asset name. For an example of creating unique asset object names by using this convention, see Creating Basic Modular Design.

`ASSET.LOAD` automatically logs a dependency between the template or element that uses the tag and the asset data that the tag retrieves.

ASSET.SCATTER (XML)	asset:scatter (JSP)

`<ASSET.SCATTER`
`NAME="assetName"`
`PREFIX="variablePrefix"/>`
	`<asset:scatter`
	`name="assetName"`
	`prefix="variablePrefix"/>`

This tag retrieves values from all of the fields of an asset object that has been retrieved (loaded) with the `ASSET.LOAD` tag and turns those values into WebCenter Sites variables. For example, to display the headline, byline, description, and so on of an article online, you can use this tag to retrieve all of those values with one call.

ASSET.GET (XML)	asset:get (JSP)

`<ASSET.GET`
`NAME="assetName"`
`FIELD="fieldName"`
`[OUTPUT="outputVariable"]/>`
	`<asset:get`
	`name="assetName"`
	`field="fieldName"`
	`[output="outputVariable"]/>`

This tag retrieves the value from one specified field of an asset object that has been retrieved (loaded) with the `ASSET.LOAD` tag and turns that value into a WebCenter Sites variable. For example, to use the headline of an article in a link to that article, use this tag to retrieve that one value.

ASSET.CHILDREN (XML)	asset:children (JSP)
`<ASSET.CHILDREN`	`<asset:children`
`NAME="assetName"`	`name="assetName"`
`LIST= "listName"`	`list="listName"`
`[CODE= "NameOfAssociation"]`	`[code="NameOfAssociation"]`
`[OBJECTTYPE= "typeOfObject"]`	`[objectype="typeOfObject"]`
`[OBJECTID="objectID"]`	`[objectid="objectID"]`
`[ORDER="nrank"]/>`	`[order="nrank"]/>`

This tag queries the AssetRelationTree table and then builds a list of assets that are children of the asset that you specified. You use this tag to retrieve assets in a collection, to retrieve the image assets associated with article assets, and so on.

Use the `RENDER.LOGDEP` tag in conjunction with `ASSET.CHILDREN` to log a dependency between the element or template in which it displays and the content that `ASSET.CHILDREN` retrieves.

16.5.9.1 Performance Notes About the Asset Tags

- `ASSET.LOAD` and `ASSET.CHILDREN` are database queries, so you should use them only when necessary, because queries to the database take time. For example, you want to include error checking code after an `ASSET.LOAD` tag and before its subsequent `ASSET.CHILDREN` tag that determines whether an asset was returned by the `ASSET.LOAD`. Don't invoke the `ASSET.CHILDREN` tag when there is no asset.

- An `ASSET.SCATTER` call takes much longer than a single `ASSET.GET` call.

16.5.10 Tags That Create Assetsets (Flex Assets)

Assetset tags specify a set of one or more flex assets that you want to retrieve from the database.

You can retrieve the following information from an assetset:

- The values for one attribute for each of the flex assets in the assetset.

- The values for multiple attributes for each of the flex assets in the assetset.

- A list of the flex assets in the assetset.

- A count of the flex assets in the assetset.

- A list of unique attribute values for an attribute for all flex assets in the assetset.

- A count of unique attribute values for an attribute for all flex assets in the assetset.

These are the assetset tags that you will use most frequently.

ASSETSET.SETASSET (XML)	assetset:setasset (JSP)
```<ASSETSET.SETASSET NAME="assetsetname" TYPE="assettype" ID="assetid" [LOCALE="localeobject"] [DEPTYPE="exact|exists|none"] />```	```<assetset:setasset name="assetsetname" type="assettype" id="assetid" [locale="localeobject"] [deptype="exact|exists|none"]/>```

ASSETSET.SETASSET builds an asset set from a single asset that you specify and defines a compositional dependency between the template or element that it displays in and the content that it retrieves.

ASSETSET.SETSEARCHEDASSETS (XML)	assetset:setsearchedassets (JSP)
```<ASSETSET.SETSEARCHEDASSETS NAME="assetsetname" [ASSETTYPES="assettype"] [CONSTRAINT="searchstateobject"] [LOCALE="localeobject"] [SITE="siteidentifier"] [DEPTYPE="exact|exists|none"]/>```	```<assetset:setsearchedassets name="assetsetname" [assettypes="assettype"] [constraint="searchstateobject"] [locale="localeobject"] [site="siteidentifier"] [deptype="exact|exists|none"]/>```

ASSETSET.SETSEARCHEDASSETS creates an assetset object which represents all assets of specific types narrowed by specified search criteria (represented by the searchstate object that you name in the constraint parameter).

This tag also defines a compositional dependency between the template or element in which it displays and the each asset in the set.

ASSETSET.GETMULTIPLEVALUES (XML)	assetset:getmultiplevalues (JSP)
```<ASSETSET.GETMULTIPLEVALUES NAME="assetsetname" LIST="listname" [BYASSET="true|false"] PREFIX="prefix"/>```	```<assetset:getmultiplevalues name="assetsetname" list="listname" [byasset="true|false"] prefix="prefix"/>```

ASSETSET.GETMULTIPLEVALUES scatters attribute values from several attributes (and potentially multiple assets) into several specified lists.

It is recommended that you use ASSETSET.GETMULTIPLEVALUES when the goal is to display a fixed-format table of assets, or to obtain many attributes of a single asset (such as for a product detail page).

ASSETSET.GETMULTIPLEVALUES has the following limitations:

- Only non-foreign attributes can be scattered.

- Text-type attributes cannot be scattered.

ASSETSET.GETATTRIBUTEVALUES (XML)	assetset:getattributevalues (JSP)

`<ASSETSET.GETATTRIBUTEVALUES`
`NAME="assetsetname"`
`ATTRIBUTE="attribname"`
`[TYPENAME="assettypename"]`
`LISTVARNAME="varname"`
`[ORDERING="ascending|`
`descending"]/>`
	`<assetset:getattributevalues`
	`name="assetsetname"`
	`attribute="attribname"`
	`[typename="assettypename"]`
	`listvarname="varname"`
	`[ordering="ascending|`
	`descending"]/>`

ASSETSET.GETATTRIBUTEVALUES gets the list of values for a specified attribute of the assets represented by an assetset.

ASSETSET.GETASSETLIST (XML)	assetset:getassetlist (JSP)

`<ASSETSET.GETASSETLIST`
`NAME="assetsetname"`
`[LIST="attriblist"]`
`[MAXCOUNT="rowcount"]`
`[METHOD="random|highest"]`
`LISTVARNAME="varname/>`
	`<assetset:getassetlist`
	`name="assetsetname"`
	`[list="attriblist"]`
	`[maxcount="rowcount"]`
	`[method="random|highest"]`
	`listvarname="varname"/>`

ASSETSET.GETASSETLIST retrieves an ordered list of assets, given optional sort criteria. The resulting list has two columns, assetid and assettype, that are sorted by the criteria that you specify.

16.5.11 Tags That Create Searchstates (Flex Assets)

Searchstate tags assemble criteria that filter the assets that you retrieve using the assetset tags.

You build a searchstate by adding or removing constraints to narrow or broaden the list of flex assets that are described by the searchstate.

These are the searchstate tags that you will use most frequently.

SEARCHSTATE.CREATE (XML)	searchstate:create (JSP)

`<SEARCHSTATE.CREATE`
`NAME="ssname"`
`[OP="and|or"]/>`
	`<searchstate:create`
	`name="ssname"`
	`[op="and|or"]/>`

SEARCHSTATE.CREATE builds an empty searchstate object. You must begin constructing a searchstate with this tag.

SEARCHSTATE.ADDSTANDARDCONSTRAINT (XML)	searchstate:addstandardconstraint (JSP)

`<SEARCHSTATE.ADDSTANDARDCONSTRAINT`
`NAME="ssname"`
`[BUCKET="bucketname"]`
`[TYPENAME="assettype"]`
`ATTRIBUTE="attribname"`
`[LIST="listname"]`
`[IMMEDIATEONLY="true|false"]`
`[CASEINSENSITIVE="true|false"]/>`
	`<searchstate:addstandardconstraint`
	`name="ssname"`
	`[bucket="bucketname"]`
	`[typename="assettype"]`
	`attribute="attribname"`
	`[list="listname"]`
	`[immediateonly="true|false"]`
	`[caseinsensitive="true|false"]/>`

SEARCHSTATE.ADDSTANDARDCONSTRAINT adds an attribute name/value constraint into a new or existing searchstate object.

You can constrain the attribute by a list of values that you specify in the list parameter.

SEARCHSTATE.ADDSIMPLESTANDARDCONSTRAINT (XML)	searchstate:addsimplestandardconstraint (JSP)

`<SEARCHSTATE.ADDSIMPLESTANDARDCONSTRAINT`
`NAME="ssname"`
`[BUCKET="bucketname"]`
`[TYPENAME="assettype"]`
`ATTRIBUTE="attribname"`
`VALUE="value"`
`[IMMEDIATEONLY="true|false"]/>`
	`<searchstate:addsimplestandardconstraint`
	`name="ssname"`
	`[bucket="bucketname"]`
	`[typename="assettype"]`
	`attribute="attribname"`
	`value="value"`
	`[immediateonly="value"]/>`

SEARCHSTATE.ADDSIMPLESTANDARDCONSTRAINT adds an attribute name/single value constraint to an existing searchstate.

This tag is the simple version of SEARCHSTATE.ADDSTANDARDCONSTRAINT. The object referred to by NAME is updated to reflect the new constraint. The new constraint replaces the old constraint if the attribute name is in the searchstate.

SEARCHSTATE.ADDRANGECONSTRAINT (XML)	searchstate:addrangeconstraint (JSP)

`<SEARCHSTATE.ADDRANGECONSTRAINT`
`NAME="ssname"`
`[BUCKET="bucketname"]`
`[TYPENAME="assettype"]`
`ATTRIBUTE="attribname"`
`LOWER="lowrange"`
`UPPER="uprange"`
`[CASEINSENSITIVE="true|false"]/>`
	`<searchstate:addrangeconstraint`
	`name="ssname"`
	`[bucket="bucketname"]`
	`[typename="assettype"]`
	`attribute="attribname"`
	`lower="lowrange"`
	`upper="uprange"`
	`[caseinsensitive="true|false"]`
	`/>`

SEARCHSTATE.ADDRANGECONSTRAINT adds a range constraint for a specific attribute name.

SEARCHSTATE.ADDRICHTEXTCONSTRAINT (XML)	searchstate:addrichtextconstraint (JSP)
```<SEARCHSTATE.ADDRICHTEXTCONSTRAINT NAME="ssname" [BUCKET="bucketname"] [TYPENAME="assettype"] ATTRIBUTE="attribname" VALUE="criteria" [PARSER="parsername"] CONFIDENCE="minlevel" [MAXCOUNT="number"] />```	```<searchstate:addrangeconstraint name="ssname" [bucket="bucketname"] [typename="assettype"] attribute="attribname" lower="lowrange" upper="uprange" [caseinsensitive="true|false"] />```

SEARCHSTATE.ADDRICHTEXTCONSTRAINT adds an attribute name and rich-text expression to the list of rich-text constraints in the searchstate.

SEARCHSTATE.TOSTRING (XML)	searchstate:tostring (JSP)
```<SEARCHSTATE.TOSTRING NAME="objname" VARNAME="varname"/>```	```<searchstate:tostring name="objname" varname="varname"/>```

SEARCHSTATE.TOSTRING converts a searchstate object into its string representation that is suitable for various uses, such as saving in a session variable or packing into a URL.

SEARCHSTATE.FROMSTRING (XML)	searchstate:fromstring (JSP)
```<SEARCHSTATE.FROMSTRING NAME="objname" VALUE="stringval"/>```	```<searchstate:fromstring name="objname" value="stringval"/>```

SEARCHSTATE.FROMSTRING provides the ability for a searchstate object to be initialized from its string representation. You must create an empty searchstate using the SEARCHSTATE.CREATE tag before you can use this tag.

16.6 About Variables Supported in WebCenter Sites

WebCenter Sites supports regular and session variables.

- Regular variables, which last for the duration of the current template or element, unless you explicitly remove them. Regular variables have a global scope.

- Session variables, which last for the duration of the current session.

WebCenter Sites provides several standard variables whose names are reserved. You can retrieve the values of these variables, but you cannot use their names for other variables that you create.

See these topics:

- [Reserved Variables](#)

- [Regular Variables](#)

- [Session Variables](#)

- [Working With Variables](#)

- [Variables and Precedence](#)

- [Best Practices with Variables](#)

16.6.1 Reserved Variables

This table defines the standard WebCenter Sites variables.

> **Note:** Modification of reserved variables is not permitted.

Table 16-11   Reserved Variables

Variable	Definition
tablename	A variable that is set to a tablename before the execsql tags can be run.
pagename	The name of the WebCenter Sites page being invoked.
ftcmd	A variable used in calls to CatalogManager.
username	A session variable that contains the name of the user who is currently logged in to the current session.
password	A session variable that contains the password of the user who is currently logged in to the current session.
authusername	A variable that you can set to the user name of a user who you want to log in to WebCenter Sites. This can be sent to WebCenter Sites using a URL.
authpassword	A variable that you can set to the password of a user who you want to log in to WebCenter Sites. This can be sent to WebCenter Sites using a URL.
currentACL	A session variable that contains the ACLs that the current user belongs to.
errno	Error numbers reported by WebCenter Sites tags.
context	Reserved for future use in the render:calltemplate tag.
site	The full name of the site, as stored in the name column of the Publication table. The site variable is set as a resarg in all of the Template and Site Entry assets. The site owns the Template and SiteEntry assets that you create within the site.
sitepfx	The site prefix (and short name of the site), as stored in the cs_prefix column of the Publication table.

Table 16-11   (Cont.) Reserved Variables

Variable	Definition
ft_ss	An internal variable that is automatically set by WebCenter Sites to support communication with Satellite Server. When ft_ss is set to true, WebCenter Sites infers that a request is from Satellite Server.
c	The asset type that a template formats. WebCenter Sites sets this variable by default when you save the Template asset.
cid	The ID of the asset being rendered or formatted by a template.
ct	The value of a child template, if there is one.
p	The ID of an asset's parent page, if there is one.
rendermode	Specifies whether a page entry is to be delivered live, exported, or previewed. By default, rendermode is live. When you use Export to Disk, or the Preview function, WebCenter Sites automatically overrides the value of this variable with export or preview. This value is used internally and must not be modified.
seid	The ID of a SiteEntry asset.
tid	The ID of a Template asset.
eid	The ID of a CSElement asset, eid is available to the CSElement's root element.

16.6.2 Regular Variables

Most of the variables that you will use while coding WebCenter Sites templates and elements are **regular variables**. Regular variables last for the duration of the current template or element, unless they are explicitly deleted using WebCenter Sites tags.

16.6.2.1 Variables with SETVAR

Inside a WebCenter Sites element, you can call the SETVAR XML or JSP tags to create a variable and establish its initial value. For example, the following SETVAR XML tag creates a variable named dog and sets its value to fido:

```
<SETVAR NAME="dog" VALUE="fido"/>
```

If the variable exists, SETVAR resets its value to the new value. For example, the following command resets the value of dog to mocha:

```
<SETVAR NAME="dog" VALUE="mocha"/>
```

16.6.2.2 Variables Using a URL

WebCenter Sites creates a page when a browser goes to a URL managed by a WebCenter Sites application. Each page is associated with a particular URL. Imagine, for example, a page associated with a URL having the following format:

```
http://host:port/servlet/ContentServer?pagename=Experiment/Hello
```

At the end of every URL, you can set one or more variables. For example, the following URL creates three variables in the `Hello` page:

```
http://host:port/servlet/ContentServer?pagename=Experiment/Hello&dog=fido&cat=fifi
```

The preceding URL creates the following variables available to `Hello`:

- A variable named `pagename` whose value is `Experiment/Hello`.

- A variable named `dog` whose initial value is `fido`.

- A variable named `cat` whose initial value is `fifi`.

16.6.2.3 Default Variables for Elements and Templates with Explorer

You can use Explorer to create default variables in a page by placing the variables in either of the following fields:

- The `resargs1` or `resargs2` fields of the `SiteCatalog` database table.

- The `resdetails1` or `resdetails2` fields of the `ElementCatalog` database table.

For example, you can use Explorer to access the `SiteCatalog` table, and then create variables `dog` and `cat` by placing name/value pairs in the `resargs1` and `resargs2` fields, as shown in this figure:

Figure 16-1   SiteCatalog Fields

Note that we placed one name/value pair in `resargs1` and another in `resargs2`. Alternatively, we could have put both name/value pairs in `resargs1`, as shown in this figure:

Figure 16-2   Values in SiteCatalog

You also can set the values of `dog` and `cat` in the `ElementCatalog` table by putting name/value pairs in the `resdetails1` and `resdetails2` fields, as shown in this figure:

Figure 16-3   ElementCatalog Fields

Variables set through the URL or through `POST` and `GET` operations take precedence over variables set using the `SiteCatalog` or `ElementCatalog` tables. For example, if a URL sets variable `dog` to `rex` and the `SiteCatalog` sets `dog` to `fido`, then the resulting value of `dog` will be `rex`.

16.6.2.4 Variables Using HTML Forms

In CGI programming, a buyer fills out a form. Then, the browser encodes the buyer's responses as name/value pairs, which get passed to the CGI script.

Although WebCenter Sites does not use traditional CGI programming, a WebCenter Sites element can still display a form. As in traditional programming, the browser encodes the buyer's responses as name/value pairs. However, instead of passing these name/value pairs to a CGI program, the pairs get passed to a different WebCenter Sites page. The receiving WebCenter Sites page can access the name/value pairs as it would access any WebCenter Sites variable.

Cookie names and values are also instantiated as variables. For more information about cookies, see About Sessions and Cookies.

16.6.3 Session Variables

HTTP is a stateless protocol. To overcome this limitation, WebCenter Sites can maintain state between requests and, thus, keep track of sessions.

A browser connection to the WebCenter Sites system establishes a session. Thereafter, the session is uniquely identified to the system. WebCenter Sites can deliver pages whose content and behavior are based on this unique identity.

When a client first enters your site, a unique session is established. WebCenter Sites associates a default user identity with a new session and maintains that information in session variables. **Session variables** contain values that are available for the duration of the session. They are saved as part of the user's session and are used to retain the value of a variable across page requests.

In a clustered configuration, the session state is maintained across all cluster members. Session variables should be used carefully, since there is a resource cost that is proportionate to the number and size of session variables used.

Session state is lost under these conditions:

- The client exits.

- The session has timed out. WebCenter Sites can optionally terminate a session if no requests have been made for some period of time.

- The application server has been restarted.

Server resources associated with the session are de-allocated when the following occurs:

- The session has been explicitly terminated by the client using a WebCenter Sites tag.

- The session has timed out.

- The application server has been restarted.

Use the `SETSSVAR` XML and JSP tags to create a session variable. If the session variable exists, `SETSSVAR` resets the variable's value. For example, the following `SETSSVAR` XML tag sets the session variable `profile` to the value `10154`:

```
<SETSSVAR NAME="profile" VALUE="10154"/>
```

16.6.4 Working With Variables

The following sections describe how to work with WebCenter Sites variables.

16.6.4.1 Syntax to Read Variables' Values

The syntax you use to read the value of a variable depends on the kind of variable:

Table 16-12   Variables and Syntax

Type of Variable	Syntax	Example
String Variable	`Variables.variable_name`	`Variables.dog`
Counter Variable	`Counters.variable_name`	`Counters.position`
Session Variables	`SessionVariables.variab le_name`	`SessionVariables.userna me`
Property	`CS.Property.property_na me`	`cs.use.short.jsp.names`

WebCenter Sites XML provides quite a few methods for accessing list variables.

16.6.4.2 Tags to Display Variables' Values

Use the `CSVAR` XML tag to display the value of any kind of variable, including properties and session variables. Use the `ics:getvar` JSP tag to view the value of a regular WebCenter Sites variable; or the `ics:getssvar` JSP tag to display the value of a session variable. For example, if the following code displays in an XML element:

```
<SETVAR NAME="mood" VALUE="happy"/>
<p>My dog is <CSVAR NAME="Variables.mood"/>.</p>
```

then the resulting page displays the following text:

```
My dog is happy.
```

You can also include literal values as part of the `NAME` argument to the `CSVAR` XML tag; for example, the following code will also generate "`My dog is happy.`", but evaluates more slowly:

```
<SETVAR NAME="mood" VALUE="happy"/>
<p><CSVAR NAME="My dog is Variables.mood"/>.</p>
```

16.6.4.3 Assigning of One Variable Value to Another Variable

You can assign the value of one variable to another variable. You accomplish this task differently if you are coding with XML than if you are coding with JSP.

JSP

If you are coding with JSP, you cannot use the `ics:getvar` tag to evaluate the variable value because you cannot nest one JSP tag within another JSP tag. To circumvent this limitation, use the `ics.GetVar` Java method to substitute variable values, as shown in the following sample code:

```
<ics:setvar name="myVar" value="Fred"/>
<ics:setvar name="yourVar" value='<%=ics.GetVar("myVar")%>'/>
<ics:getvar name="yourVar"/>
```

> **Note:**
>
> You must enclose the expression that evaluates the variable value ('`<%=icsGetVar("myVar")%>`' in the example) in single quotes. Otherwise your JSP element will throw an exception.

XML

The following lines of XML assign the value `carambola` to a variable named `your_favorite`:

```
<SETVAR NAME="my_favorite" VALUE="carambola"/>
<SETVAR NAME="your_favorite" VALUE="Variables.my_favorite"/>
```

Taking this one step further, you can concatenate two variable values and assign the result to a third variable. For example, the following sets the variable `car` to the value `red rabbit`.

```
<SETVAR NAME="color" VALUE="red"/>
<SETVAR NAME="model" VALUE="rabbit"/>
<SETVAR NAME="car" VALUE="Variables.color Variables.model"/>
```

16.6.4.4 Variables in HTML Tags

You can use XML and JSP variables within traditional HTML tags, although you code differently to accomplish this in XML and JSP.

JSP

If you are coding with JSP, you use the `ics:getvar` tag or the `ics.GetVar` Java method to evaluate the variable value.

You can also use the `ics.resolvevariables` tag to resolve variables that are contained within a string. For example, the following code displays the phrase, "The date is," along with the value of the `CS.Date` variable:

```
<ics.resolvevariables name="The date is $(CS.Date)." delimited="true"/>
```

The `delimited` parameter indicates that you have used the delimiters `$(` and `)` to explicitly mark the variable or variables that you want to resolve. To use variables to specify a list name and a column in that list, for example, you use the following syntax:

```
<ics.resolvevariables name="$(Variables.listname).$(Variables.columnname)"
delimited="true"/>
```

If the `delimited` parameter is set to `false`, no delimiters are used to set off variables.

XML

You can use XML variables inside HTML tags if you use the appropriate attributes. For example, the following code does not contain the appropriate attributes and, therefore, does not set the background color to red:

```
<SETVAR NAME="color" VALUE="red"/>
<TABLE bgcolor="Variables.color">
...
```

To use XML variable values within an HTML tag, you must use the `REPLACEALL` attribute within that HTML tag. The `REPLACEALL` attribute tells the system to substitute the current value of this XML variable within this HTML tag. Therefore, the correct way to code the preceding lines is as follows:

```
<SETVAR NAME="color" VALUE="red"/>
<TABLE bgcolor="Variables.color" REPLACEALL="Variables.color">
...
```

You can combine multiple variable values within one `REPLACEALL` attribute. For example, the following HTML `TABLE` tag uses two XML variables:

```
<SETVAR NAME="color" VALUE="red"/>
<SETVAR NAME="myborder" VALUE="3"/>
<TABLE bgcolor="Variables.color" border="Variables.myborder"
    REPLACEALL="Variables.color,Variables.myborder">
...
```

The `<REPLACEALL>` tag is an alternative to the `REPLACEALL` attribute. The `<REPLACEALL>` tag performs substitutions within its domain; for example:

```
<SETVAR NAME="highlight" VALUE="red"/>
<SETVAR NAME="diminish"  VALUE="gray"/>
<REPLACEALL LIST="Variables.highlight,Variables.diminish">
<TABLE>
  <TR BGCOLOR="Variables.highlight"><TD>Diamonds</TD></TR>
  <TR BGCOLOR="Variables.highlight"><TD>Pearls</TD></TR>
  <TR><TD>Malachite</TD></TR>
  <TR BGCOLOR="Variables.diminish"><TD>Coal</TD></TR>
</TABLE>
</REPLACEALL>
```

This figure shows the output of this section:

Figure 16-4   Sample Output
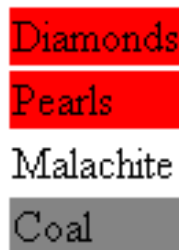
The `REPLACEALL` tag performs a string search and replace, and is, therefore, potentially very slow. Use the `REPLACEALL` attribute where possible. If you must use the `REPLACEALL` tag, keep the amount of code you enclose with it as small as possible.

16.6.4.5 Evaluation of Variables with IF/THEN/ELSE

WebCenter Sites XML and JSP provides the `IF/THEN/ELSE` construct available in most computer languages. However, the only conditional operation for variables is to compare two values for equality or inequality. You can't, for example, compare two values to see if one is greater than another. (You can write Java code to do that, however.)

For example, the following code branches depending on the value of a variable named `greeting`.

```
<IF COND="Variables.greeting=Hello">
<THEN>
    <p>Welcome.</p>
</THEN>
<ELSE>
    <p>So long.</p>
</ELSE>
</IF>
```

If greeting is set to `Hello`, then WebCenter Sites generates the HTML:

```
<p>Welcome.</p>
```

If greeting is set to anything other than `Hello`, WebCenter Sites generates:

```
<p>So long.</p>
```

16.6.5 Variables and Precedence

Variables set through a URL or through HTTP `GET` and `POST` operations take precedence over variables set with the `resargs` and `resdetails` columns in the `SiteCatalog` and `ElementCatalog` tables.

16.6.6 Best Practices with Variables

Because all variables are global and the syntax for accessing variables from items in lists and from other sources is the same, good coding practices help you to avoid errors. For example:

- Because it is easy to reuse base names in your elements, use prefixes in front of variables to define them uniquely. The recommended syntax to use is: `Variables.assettype:fieldname`.

  For example, `Variables.Article:description`.

  The `ASSET.SCATTER` tag makes it easy for you to use this syntax through its `PREFIX` attribute. For more information about this tag, see Coding Elements for Templates and CSElements.

- If you are going to use the `RESOLVEVARIABLES` tags to resolve your variables, set the `DELIMITED` parameter to `true` and use the delimiters `$(` and `)` to explicitly indicate the variables you want to resolve.

- Use debugging to catch naming conflicts. Set the `com.fatwire.logging.cs` property in the `loggingconfig.xml`. When this property is enabled, WebCenter Sites writes a record of all the variables that are created to the WebCenter Sites log file.

For a list of the error values that WebCenter Sites tags can write to the `errno` variable, see the *Oracle Fusion Middleware Tag Reference for Oracle WebCenter Sites Reference*.

16.7 Other WebCenter Sites Storage Constructs

WebCenter Sites supports several storage constructs such as built-in, lists, and counters.

See these topics for what these constructs are and how you can define them:

- Built-ins

- Lists

- Counters

16.7.1 Built-ins

WebCenter Sites provides several built-ins, which return values such as the current date.

The general syntax of a built-in is:

```
CS.builtin
```

For example, `UniqueID` is a built-in that generates a unique ID. The following syntax generates or references this built-in variable:

```
CS.UniqueID
```

For a list of built-ins in WebCenter Sites, see *Oracle Fusion Middleware Tag Reference for Oracle WebCenter Sites Reference*.

16.7.2 Lists

A list consists of a table of values organized in rows and columns. Use the `SETROW` or `GOTOROW` tags to identity the proper row.

The following entities create lists:

- The `SELECTTO`, `EXECSQL`, `CATALOGDEF`, `STRINGLIST` and `CALLSQL` tags.

- The `CatalogManager` commands.

- The `TreeManager` commands.

- The `Custom` tags.

Use the following syntax to refer to a current row's column value:

```
listname.colname
```

For example, if a list named `cars` had a column named `color`, the value of the current row would be referenced as:

```
cars.color
```

16.7.2.1 Looping Through Lists

Use the `LOOP` XML tag or the `ics:listloop` JSP tag to iterate through a list. For each row in the list, WebCenter Sites executes the instructions between the loop tags.

For example, consider a table named `MyCars` containing the following rows:

Table 16-13   Example Table with Rows

id	Model	Color	Year
224	Ford Focus	blue	2001
358	VW Rabbit	red	1998
359	Toyota Corolla	yellow	2000
372	Alpha Romeo Spider	red	1982
401	Porsche 911	red	1984
423	Dodge Voyager	tan	1991

The following XML searches `MyCars` for red cars. The `SELECTTO` XML and JSP tags write this information into a list variable named `carlist`.

```
<SETVAR NAME="color" VALUE="red"/>
<SELECTTO FROM="MyCars" WHERE="color" WHAT="*" LIST="carlist"/>
Red cars: <BR/>
<OL>
```

```
<LOOP LIST="carlist">
    <LI><CSVAR NAME="carlist.model"/> </LI>
</LOOP>
</OL>
```

The preceding XML generates the following HTML:

```
Red cars: <BR/>
<OL>
 <LI> VW Rabbit </LI>
 <LI> Alpha Romeo Spider </LI>
 <LI> Porsche 911 </LI>
</OL>
```

16.7.3 Counters

A counter is an XML variable whose value is an integer. Three tags control counters:

Table 16-14   Counters

Tag	What It Does
SETCOUNTER	Initializes a counter variable
INCCOUNTER	Changes the counter's value by a specified amount
REMOVECOUNTER	Destroys the counter variable

To create a counter, you call SETCOUNTER. To change its value, call INCCOUNTER. For example, consider the following code:

```
<SETCOUNTER NAME="c" VALUE="10"/>
<INCCOUNTER NAME="c" VALUE="3"/>
<p>Current value is <CSVAR NAME="Counters.c"/></p>
```

The output of this code is:

```
Current value is 13
```

Notice that you reference counter variables using the syntax:

```
Counters.name
```

16.8 About Values for Special Characters

Use the hexadecimal character representation of special (non-alphanumeric) characters in your XML or JSP instead of the special characters themselves. For example, to specify a space as part of a variable value, you can use this line: `<SETVAR NAME="foo" VALUE="foo%20bar"/>`

Here are hexadecimal values for special characters that are commonly used in WebCenter Sites:

Table 16-15   Values for Special Characters

Hexadecimal Value	Character
%22	doublequote (")
%20	one space

Table 16-15   (Cont.) Values for Special Characters

Hexadecimal Value	Character
%3c	less than sign (<)
%3e	greater than sign (>)
%26	ampersand (&)
%09	tab (\t)
%0a	newline (\n)
%0d	carriage return (\r)
%25	percent (%)

17

About Sessions and Cookies

Sessions store information about visitors, and cookies store information about visitors that lasts between sessions. See examples and tips that can help you manage sessions and cookies efficiently.

Topics:

- About Sessions
- Session Lifetime
- Sessions Example
- About Cookies
- Cookie Example
- Tips and Tricks
- Satellite Server Session Tracking

17.1 About Sessions

Sessions are necessary when a web server managing a website uses HTTP stateless protocol that doesn't require the server to retain the session information. The application server starts a session as soon as a visitor begins browsing through pages. The browsing information gets stored in session variables.

Imagine a website containing two pages: `main` and `water`. Suppose a visitor sees `main` first and then moves on to `water`. So, if a typical web server is managing this site, any knowledge gathered at `main` is lost when the visitor browses over to `water`. In other words, `water` cannot take advantage of any information that the visitor might have provided at `main`.

To get around this limitation, application servers detect when a visitor first enters a website. At that point, the application server starts a *session* for this visitor. In the preceding example, when the visitor requests the `main` page, the application server starts a session. The website designer can use `main` to gather information about the visitor and store that information in *session variables*. The information in session variables is available to all subsequent pages. So, for example, if Bob provides his age to `main`, and `main`'s designer wrote the age to a session variable, then `water` could easily access Bob's age.

Session variables contain values available for the duration of the session. When the session ends, the application server destroys the session variables associated with that session. Each session variable consumes memory on the application server, so creating unnecessary session variables can hurt performance.

WebCenter Sites automatically creates some session variables; the website developer can optionally create others. The application server can maintain sessions on a cluster.

17.2 Session Lifetime

A session begins when a visitor first visits your website. It ends when the visitor terminates his browser, the system administrator stops the application server, or the session has timed out.

The `cs.timeout` property is used by WebCenter Sites to set the session timeout value in the application server. If this property is set to `300`, then a user session becomes invalid in 300 seconds, or 5 minutes.

See these topics:

- Session Variables Maintained by WebCenter Sites

- Logging In and Logging Out

17.2.1 Session Variables Maintained by WebCenter Sites

Upon creating a session, WebCenter Sites automatically creates the session variables described in this table:

Table 17-1   Session Variables

Session Variable	What it Holds
`SessionVariables.currentUser`	The `id` of the visitor logged in.
`SessionVariables.currentAcl`	The comma-separated list of all ACLs to which this visitor belongs. If the visitor has not explicitly logged in, the default ACL is `Browser`.
`SessionVariables.username`	The user name under which this visitor is logged in. If the visitor has not explicitly logged in, the default user name is `DefaultReader`.
`SessionVariables.iniFile`	The name of the file containing WebCenter Sites properties.

17.2.2 Logging In and Logging Out

When a visitor first visits the site, WebCenter Sites creates a session and implicitly logs in the visitor as `DefaultReader`. During the session, if the visitor explicitly logs in, WebCenter Sites automatically updates the values of `SessionVariables.currentUser`, `SessionVariables.currentAcl`, and `SessionVariables.username`. Logging in does not affect the values of any other session variables. In other words, if your pages create session variables before a login, then those values are still valid after the login. When a visitor explicitly logs out, the WebCenter Sites-generated session variables automatically revert to the values they held before login. For example, consider the following sequence:

1. A visitor first visits a page, so the value of `SessionVariables.username` is `DefaultReader`.

2. The visitor logs in as marilyn, so the value of `SessionVariables.username` is `marilyn`.

3. If marilyn logs out, the value of `SessionVariables.username` reverts to `DefaultReader`.

To trigger a logout, you call the `<CATALOGMANAGER>` tag with the `ftcmd=logout` modifier. When issuing this tag, you can optionally supply the `killsession` modifier, which destroys the current session. You can then create a new session by invoking the `<CATALOGMANAGER>` tag with the `ftcmd=login` modifier.

17.3 Sessions Example

A simple session example with a few elements can help you understand how sessions work.

Here's an example, consisting of three very short elements:

Table 17-2   Sessions Example

Element	What it Does
FeelingsForm	Asks visitors to pick their current mood.
SetFeelings	Assigns the current mood to a session variable.
Meat	Evaluates the session variable.

See these topics:

- FeelingsForm Element

- SetFeeling Element

- Meat Element

17.3.1 FeelingsForm Element

The feelings form does not really involve sessions or variables. This element merely generates a form. The visitor's chosen mood is passed to the `SetFeeling` element:

```
<form action="ContentServer" method="post">
   <input type="hidden" name="pagename"
          value="CSGuide/Sessions/SetFeelings"/>

 <P>How are you feeling right now?</P>
 <P>
 <select name="Feeling" size="1">
    <option>Good</option>
    <option>Not so Good</option>
 </select>
 </P>

 <P><input type="submit" name="doit" value="Submit"/></P>
</form>
```

The resulting page looks like this:

Figure 17-1   Sample Page

17.3.2 SetFeeling Element

Upon clicking the Submit button, the visitor is transported to `SetFeeling`. This element assigns the visitor's mood to a new session variable named `CurrentFeeling`.

```
<SETSSVAR NAME="CurrentFeeling" VALUE="Variables.Feeling"/>

<P>Welcome to our site.</P>

<P>Now proceed to
<A href="ContentServer?pagename=CSGuide/Sessions/Meat">
some meaty content.
</A></P>
```

The resulting page looks as follows:

```
Welcome to our site.
Now proceed to some meaty content.
```

If an element in this application asked the visitor to login, WebCenter Sites would have automatically set the `username` session variable to the visitor's login name. In that case, you could have personalized the welcome message in `SetFeeling` as follows:

```
<P>Welcome to our site, <CSVAR NAME="SessionVariables.username"/>
</P>
```

17.3.3 Meat Element

Upon clicking some meaty content, the visitor is transported to the `Meat` page. This page evaluates the session variable:

```
<IF COND="SessionVariables.CurrentFeeling=Good">
 <THEN>
     <P>Sessions are happiness.</P>
 </THEN>
 <ELSE>
     <P>Don't let sessions get you down.</P>
 </ELSE>
</IF>
```

A visitor in a not so good mood sees:

```
Don't let sessions get you down.
```

Notice how `CurrentFeeling` was available to `Meat`. In fact, `CurrentFeeling` is available to any other elements in the session.

17.4 About Cookies

A *cookie* is a string that your application writes to the visitor's browser. A cookie stores information about visitors that lasts between sessions. The visitor's browser writes this string to a special cookie file on the visitor's disk. When that visitor returns to your website, the visitor's browser sends a copy of the cookie back to the web server that set it. Once a cookie has been created, it is available as a variable to elements on a page.

For example, your application might store the visitor's favorite sports team in a cookie. Then, when the visitor returns, your application could retrieve the cookie and use its information to display the team logo in a banner. When cookies are no longer needed, you can delete them.

This section includes the following topics:

- CookieServer

- Cookie Tags

17.4.1 CookieServer

CookieServer is a servlet that sets cookies for you. You access CookieServer by creating cookies with the `satellite.cookie` tag.

17.4.2 Cookie Tags

WebCenter Sites offers two tags for managing cookies:

Table 17-3   Cookie Tags

Tag	Use
satellite.cookie	Sets a cookie on the client's browser.
REMOVECOOKIE	Deletes a cookie from the client's browser.

There is no special tag to obtain the value of a cookie. Instead, when a visitor returns to the website, WebCenter Sites loads the value of the cookie as a regular variable.

When creating a cookie (by calling `satellite.cookie`), you can specify the following attributes:

Table 17-4   Cookie Attributes

Attribute	Value
name	Name of the cookie. This also serves as the name of the incoming variable containing the value of the cookie. **Note**: Cookies in the WebCenter Sites page context are treated as variables. Therefore, when a cookie and an asset attribute share the same name, they are treated as the same variable.
expiration	Time in seconds after which the cookie no longer is sent to the web server.
security	Optionally set security on the cookie.
URL	Restrict that the cookie only be sent on this URL.

Table 17-4   (Cont.) Cookie Attributes

Attribute	Value
Domain	Restrict that the cookie only be sent to URLs in the specified domain.

Because they feel that cookies are a security threat, some visitors configure their browsers to reject cookies. If the information in the cookie is critical, your application must be prepared for this.

You must set or remove cookies before using any tags that stream content back to the visitor's browser. You must set or remove cookies even before the <HTML> tag.

17.5 Cookie Example

A cookie example with several short elements can help you understand how cookies work.

Here is an example:

Table 17-5   Cookie Elements

Element	What it Does
Start	Determines whether a cookie is set. If cookie is set, call DisplayWelcome. If cookie is not set, call GetColorPreference.
ColorForm	Displays a form that asks visitor to pick her favorite color.
CreateCookie	Creates a cookie on this visitor's browser. Then, redirects visitor to DisplayWelcome.
DisplayWelcome	Displays a simple welcome message in the visitor's favorite color.

See these topics:

- Start.xml

- ColorForm

- CreateCookie

- DisplayWelcome

- Running the Cookie Example

17.5.1 Start.xml

The Start.xml element determines whether the cookie has been set. If the cookie has been set, WebCenter Sites stores its value inside a regular variable named Variables.ColorCookie. The code for Start.xml is as follows:

```
<IF COND="IsVariable.ColorCookie=true">
 <THEN>
    <CALLELEMENT NAME="CSGuide/Sessions/DisplayWelcome"/>
 </THEN>
```

```
 <ELSE>
    <CALLELEMENT NAME="CSGuide/Sessions/ColorForm"/>
 </ELSE>
</IF>
```

17.5.2 ColorForm

The `ColorForm.xml` element displays an HTML form to gather the visitor's favorite color. The code for `ColorForm.xml` is as follows:

```
<form action="ContentServer" method="post">
   <input type="hidden" name="pagename" value="CSGuide/Sessions/CreateCookie"/>

 <P>What is your favorite color?</P>
 <P>
 <select name="FavoriteColor" size="1">
    <option>Red</option>
    <option>Green</option>
    <option>Blue</option>
 </select>
 </P>

<P><input type="submit" name="doit" value="Submit"/></P>
</form>
```

17.5.3 CreateCookie

The `CreateCookie.xml` element sends a cookie named `ColorCookie` to the visitor's browser. If the visitor has disabled cookies, the browser ignores the request to set a cookie. If the visitor has enabled cookies (the default), the browser writes the cookie to this system's cookie file.

The following is the code for `CreateCookie.xml`:

```
<satellite.cookie NAME="ColorCookie" VALUE="Variables.FavoriteColor"
TIMEOUT="31536000" SECURE="false"/>

<CALLELEMENT NAME="CSGuide/Sessions/DisplayWelcome"/>
```

The preceding code sets the value of the cookie to the visitor's favorite color. This cookie lasts for one year (31,536,000 seconds).

17.5.4 DisplayWelcome

By the time DisplayWelcome is called, the cookie has been set. The following code uses the value of the cookie to display a welcome message in the visitor's favorite color:

```
<H1><font color="Variables.ColorCookie"
     REPLACEALL="Variables.ColorCookie">
Displaying a Friendly Welcome.
</font></H1>
```

17.5.5 Running the Cookie Example

To run the cookie example, use your browser to go to the following pagename: `CSGuide/Sessions/Start`.

The first time you run this example, all four elements execute. After the first time, only `Start` and `DisplayWelcome` execute.

17.6 Tips and Tricks

Some tips and tricks about using sessions can help you improve visitors' browsing experience while efficiently using the system resources.

The following suggestions might be useful:

- In a cluster, session state must be replicated across cluster members. In a cluster, try to keep session size to a minimum; don't store more than 2 Kilobytes of session data per client.

- Determine reasonable session timeout values. Setting timeouts that are too large tie up system resources. Setting them too small forces visitors to log in with annoying frequency.

17.7 Satellite Server Session Tracking

Websites that present personalized content to visitors must track sessions. WebCenter Sites and Satellite Server both track sessions and set two cookies in the visitor's browser. Each cookie independently tracks a session. This redundancy is useful to maintain aWebCenter Sites session when a Satellite Server goes down.

Though Satellite Server will only serve session-specific pagelets back to the person who originally requested them, explicitly flushing session-specific information about user logout is a wise way to conserve space in the Satellite Server cache.

The following sections describe how to flush session information from Satellite Server:

- Flushing a Session Using a URL

- Flushing Current Session Information

- Flushing Other Session Information

17.7.1 Flushing a Session Using a URL

You can flush all data pertaining to a particular session. To do this, from WebCenter Sites post a form to a URL in the following format:

```
https://host:port/servlet/FlushServer?
reset=true&username=username&password=password&ssid=sessionID
```

where:

Table 17-6   Session Parameters

Parameter	Value
host	Specify the name of the Satellite Server host whose cache is to be flushed.
port	Specify 80 (the default) unless you re-configured Resin to run on a different port.
username	Specify the value assigned to the user name property.
password	Use the value assigned to the password property.

Table 17-6   (Cont.) Session Parameters

Parameter	Value
sessionID	Specify the session ID (the one maintained by WebCenter Sites, and not by Satellite Server) representing the session to be removed.

17.7.2 Flushing Current Session Information

To flush the information for the Satellite Server session that you are currently in, use the `FlushServer` URL with the current session's ID. The current session ID (`ssid`) is stored in a session variable with a name that is dependent upon your application server. You can see this name by looking at the session variable `HTTP_COOKIE`.

- Use the following Java code to flush the information for the current session:

```
String value;
String name = "WebLogicSession";
value = ics.GetVar(name);

String sFlushSessionUrl ="http://mysatellite:80/servlet/
FlushServer?username=ftuser&password=ftuser&
reset=true&ssid=" + value;"

String sSatTest1Results = Utilities.readURL(sFlushSessionUrl);
```

17.7.3 Flushing Other Session Information

To flush information from a session other than the one you are in follow these steps:

1. Add the following tag to the container page that contains the pagelets that you want to flush:

```
<satellite.page
pagename="QA/Satellite/Functional/xml/pagelet4"
cachecontrol="session:0:00:00 */*/*"/>
```

The `cachecontrol` value of `"session:0:00:00 */*/*"` means that every session that requests this page creates a pagelet that can only be viewed by subsequent requests by that session. Once the session for a given page expires, that page cannot be viewed again. The container page will expire from the cache at midnight each day.

2. After setting the `cachecontrol` parameter for the container page, use the Inventory servlet with the keys parameter to get its session ID (`ssid`). The `ssid` is the string that precedes the protocol and server name. For example, if the Inventory servlet displays:

```
OuCOTrh9yporWfgu8Uthttp://myserver:80/servlet/
ContentServer?pagename=QA/Satellite/Functional/xml/pagelet4
```

then the `ssid` is `OuCOTrh9yporWfgu8U`.

3. Flush information from the session by using the `ssid` you found with the FlushServer URL. For example:

```
http://myserver:80/servlet/
FlushServer?username=ftuser&password=ftuser&reset=true&
ssid=OuCOTrh9yporWfgu8U
```

> **Note:**
>
> To flush information from a session other than the one you are in, you should have session affinity enabled.

18

Creating Template, CSElement, and SiteEntry Assets

Template, CSElements, and SiteEntry are rendering assets. These assets wrap content assets in HTML, CSS, JSS and render them in the browser. CSElement asset represents an element, SiteEntry represents the name of a page, and Template represents both, the element and the page name. WebCenter Sites uses these assets to generate website pages.

Topics:

- About Template, CSElement, and SiteEntry Assets

- About Pages

- Using CSElement, Template, and SiteEntry Assets

- Creating Template Assets

- Creating CSElement Assets

- Creating SiteEntry Assets

- Managing Template, CSElement, and SiteEntry Assets

- Using Oracle WebCenter Sites Explorer to Create and Edit Element Logic

See Coding Elements for Templates and CSElements.

Note: When creating templates for use with Visitor Services, you can use the Visitor Service Helper class to get the current visitor's profile information and to manage visitor cookie lifecycles. See Linking Visitor Profiles and Managing Cookies.

In WebCenter Sites, template developers must ensure that the template code hits Visitor Services (using the Visitors Client API or WebCenter Sites Wrapper API, that is, class `VisitorServiceHelper.java`) only when the Visitor Services application is correctly configured and running, or Content Server errors can occur.

18.1 About Template, CSElement, and SiteEntry Assets

With Template, CSElements, and SiteEntry assets you build pagelets and elements to develop your online sites. These assets represent page names and elements that WebCenter Sites uses to generate pages. When you create a CSElement asset, you code an element. When you create a SiteEntry asset, you name a page. When you create a template, you do both: you code an element and you name a page.

Here are some important points that can help you use these asset types effectively:

- Template assets are classified as typed or typeless depending on whether they apply to a single asset type or no asset type.

- If you are using SiteLauncher (to replicate sites or share Template and CSElement assets), WebCenter Sites requires element logic to indirectly refer to assets, asset types, attribute names, and template names. To this end, the WebCenter Sites interface introduces the Map screen (for example, Configure the Map); the API introduces the `render:lookup` tag.

  Using the Map screen, you assign an alias to each value. You can then hardcode the aliases in the element logic and use the `render:lookup` tag to retrieve the actual values from the aliases at runtime.

- The **Cache Rules** field has been simplified to reduce errors. Template developers can choose cached, uncached, or advanced. Selecting **Advanced** allows developers to set caching rules individually for WebCenter Sites and Satellite Server.

- A new tag, `calltemplate`, was introduced to invoke templates in a way that simplifies the template writing process.

- The **PageCriteria** field has been renamed to **Cache Criteria**. It accepts the following reserved parameters: c, cid, context, p, rendermode, site, sitepfx, ft_ss, and custom-defined parameters.

  Cache criteria values are stored in the `pagecriteria` column of the `SiteCatalog` table (in previous versions they were stored in the `resargs` columns of the `SiteCatalog` table).

  The **Cache Criteria** field is also used to hold variables that enable the **Extra Parameters** section in the CKEditor and make them available to users, in the **Include asset link** and **Add asset link** dialog boxes. The **Extra Parameters** section provides a way of passing custom parameters (such as image dimensions) to the template. See step 2 in Configure SiteEntry about extra parameters. See *Oracle Fusion Middleware Using Oracle WebCenter Sites*.

- Forms for creating Template and CSElement assets have been subdivided by tabs; fields are organized by function on the tabs.

18.2 About Pages

In the WebCenter Sites context, an online page is the composition of several components into a viewable, final output. Creating that output is called **rendering**. (Making either that output or the content that is to be rendered available to the visitors on your public site is called publishing.)

WebCenter Sites renders pages by executing the code associated with page names. The name of a page is passed to WebCenter Sites from a browser and WebCenter Sites invokes the code associated with that page name. The code is actually a named file, a separate chunk of code called an element.

The code in your elements identifies and then loads assets to display in those pages or pagelets, and passes other page names and element names to WebCenter Sites. When WebCenter Sites invokes an element, all of the code in the element is executed. If there are calls to other elements, those elements are invoked in turn. Then the results, the images, articles, linksets, and so on, including any HTML tags, are rendered into HTML code (or some other output format if your system is configured to do so).

Template, CSElement, and SiteEntry assets represent elements and pagelets as follows:

- A CSElement asset is an element.

- A SiteEntry asset is the name of a page or a pagelet.

- A Template asset is both an element and a page or pagelet that renders an asset.

See these topics:

- [Elements, Pagelets, and Caching](#)

- [Calling Pages and Elements](#)

- [Page vs. Pagelet](#)

18.2.1 Elements, Pagelets, and Caching

Pages and pagelets are cacheable. They have cache criteria set for them that determines whether they are cached and, if so, for how long.

Elements do not have cache criteria. When your code calls an element directly by name, without going through a page name, the output is displayed in the page that called the element's name and that output is cached as a part of that page.

To cache the output from an element separately from the output of the page that called it, you must provide a page name for it and call it by its page name. The code in a Template asset has a page name by default. To provide a page name for a CSElement asset, you create a SiteEntry asset and select the CSElement asset for it.

18.2.2 Calling Pages and Elements

To see a WebCenter Sites page, you provide a URL that includes the name of the page. A WebCenter Sites URL looks like this:

For WebLogic and WebSphere:

```
http://host:port/servlet_context_path/ContentServer?pagename=name_of_page
```

where:

- `host` is the name of the server that is hosting the WebCenter Sites system,

- `port` is the port number of the web server,

- `servlet_context_path` is the path that the application server gives to the WebCenter Sites web application, and

- `name_of_page` is the page name.

This syntax passes the name of a page to the ContentServer servlet, which then renders the page.

For example, to see the home page of a site, you would enter a URL like this:

```
http://site-address/servlet/ContentServer?pagename=
site-name/Page/Home
```

When you code your elements, you use tags that programmatically call the pagelets and elements that you want to display in your site. These tags pass the names of pages and elements to the ContentServer servlet just as a URL entered in a browser passes a page name to the ContentServer servlet.

To call a page name, use the `render:satellitepage` (`RENDER.SATELLITEPAGE`) tag. For example:

```
<render:satellitepage pagename="site-name/Page/Home" />
```

To call an element directly by name, use the `render:callelement` (`RENDER.CALLELEMENT`) tag. For example:

```
<render:callelement elementname="site-name/Common/TextOnlyLink" />
```

To call a template by name, use the `render:calltemplate` tag. For example:

```
<render:calltemplate
        site='<%=ics.GetVar("site")%>'
        slotname="Head"
        tid='<%=ics.GetVar("tid")%>'
        c='<%=ics.GetVar("c")%>'
        cid='<%=ics.GetVar("cid")%>'
        tname='<%=ics.GetVar("HeadVar")%>'>
    <render:argument name="p" value='<%=ics.GetVar("p")%>' />
</render:calltemplate>
```

Note:

When you use Oracle WebCenter Sites Explorer to examine `SiteCatalog` and `ElementCatalog` entries, they are presented as folders and subfolders that visually organize the pages and pagelets.

However, these entries are simply rows in a database table (there is no actual hierarchy). Therefore your code must always call a page entry or an element entry by its entire name. You cannot use a relative path.

Your code calls template, CSElement, and SiteEntry assets as follows:

- Because a SiteEntry is a pagelet, you use the `render:satellitepage` tag to call SiteEntry assets from within your element code.

- Because a CSElement is an element, you use the `render:callelement` tag to call CSElement assets from within your element code.

- Because a template is both an element and a page name, you can use either of the above, although typically the `render:calltemplate` tag is designed to be used for templates. It encapsulates the functionality of `render:satellitepage` and `render:callelement` and other features, such as parameter validation.

18.2.3 Page vs. Pagelet

The table below lists the various terms that include the word page and defines them in the context of their usage in the documentation for WebCenter Sites, the WebCenter Sites modules, and the products.

Table 18-1   Page Vs. Pagelet

Term	Definition
pagelet	The results of an HTTP request displayed in a browser as one piece of a rendered page. It has an associated element file.
	A pagelet can be cached in the WebCenter Sites and Satellite Server page caches.
page	The results of an HTTP request displayed in a browser window. A page is created by compiling several parts of pages (pagelets) into one final, displayed or rendered page. It has an associated element file.
	A page can be cached in the WebCenter Sites and Satellite Server page caches.
page name	The complete name of a page or pagelet. For example: Developer's Samples/Home/Rendering API/Asset Reader.
page asset	Page assets do **not** represent page names. They represent logical containers for content. These containers can be arranged into a tree structure for navigation of site content.
	You create page assets and then place them in position in the Site Navigation tree which is visible on the **Site** tab in the tree in the left pane of the WebCenter Sites interface. You associate other content and site design assets with them and then you publish them.

18.3 Using CSElement, Template, and SiteEntry Assets

Template assets render content assets into pages and pagelets, CSElement assets reuse content assets in multiple places, or call them from multiple types of templates, or both. SiteEntry assets accompany CSElement assets when the output from CSElements should be cached as separate pagelets.

You can use the**CSElement** or **Template** asset forms available in the Admin interface to create CSElement and Template assets.

Note:

Elements for Template assets and CSElements can be coded in Oracle WebCenter Sites Explorer. However, the procedure is not recommended for reasons dealing mostly with compositional dependencies and updates to the cache. Developers who prefer to use Oracle WebCenter Sites Explorer must follow the steps in Using Oracle WebCenter Sites Explorer to Create and Edit Element Logic to ensure the validity of the Template or CSElement assets.

Because page names and elements are assets, you can manage your code and page names in the same way you manage your content: you can use workflow, revision tracking, approval, and preview, and the Mirror to Server publishing method to move your code and page names to the management and delivery systems.

> **Note:**
>
> **Revision tracking** Never use the revision tracking feature in the Oracle WebCenter Sites Explorer tool to enable revision tracking directly on the SiteCatalog or ElementCatalog tables.
>
> **Mirror to Server** If templates or CSElements refer to elements that are not associated with a template or CSElement asset, these elements are not automatically mirrored to the publishing destination. You must move them manually with the CatalogMover utility. For this reason, we do not recommend using elements that are not wrapped by CSElements.

See these topics:

- Template Assets

- CSElement Assets

- SiteEntry Assets

- Non-Asset Elements

18.3.1 Template Assets

Templates render other assets into pages and pagelets. This in turn creates the look and feel of your online site. You create a standard set of templates for each asset type, except CSElement and SiteEntry assets, so that all assets of the same type are formatted in the same way.

This process allows content providers to preview their content by selecting formatting code for the content, but not requiring them to code themselves or allowing them to change your standard, approved code.

When you save a Template asset, WebCenter Sites does the following:

- Creates a row in the `Template` table for the asset.

- Creates an element entry in the `ElementCatalog` table. The name of the entry uses the following convention:

  AssetTypeName/TemplateName

  where:

  – *AssetTypeName* is the asset type formatted by the Template asset and element.

  – *TemplateName* is the name of the template.

- Creates a page entry in the SiteCatalog table. The name of the page entry uses the following convention:

  SiteName/AssetTypeName/TemplateName

  where:

  – *SiteName* is the name of the site that the template belongs to, which is the site that you were working in when you created the template. WebCenter

Sites obtains this name from the Publication table. (In previous versions of the product, sites were called publications.)

– *AssetTypeName* is the asset type formatted by the Template asset and element

– *TemplateName* is the name of the template.

Note:

Do not change the name of the page entry that WebCenter Sites creates.

- Creates new rows in other tables that support the operation of the Template asset. The tables start with the name: `Template_`

- Creates a new row in the AssetPublication table to associate your template with your site.

18.3.2 CSElement Assets

You use CSElement assets for the following kinds of things:

- Code that is not for rendering an asset and that you want to reuse in multiple places or call from multiple types of template or both. For example, you have six templates that use the same top banner so you create a CSElement asset for the code in the banner and call that element from each template. This way, if you decide to change the way the banner works, you only have to change it in one place.

- Recommendations for Oracle WebCenter Sites: Engage. If you create a dynamic list recommendation, you must create a CSElement asset to build the dynamic list. See Understanding Recommendation Assets. These assets do not render content, but exist for logic processing.

When you save a CSElement, WebCenter Sites does the following:

- Creates a row in the `CSElement` table for the asset.

- If you have coded the element in the CSElement form, creates an element entry in the ElementCatalog table. The name of the entry is the name that you entered into the ElementCatalog Entry Name field in the form.

- Creates a new row in the `AssetPublication` table to associate your template with your site.

18.3.3 SiteEntry Assets

You use SiteEntry assets for the following kinds of things:

- If you are using the CS-Designer tool, you use SiteEntry assets to represent code snippets. In that interface, when you drag and drop a code snippet into a page, you are dropping in a WebCenter Sites call to a page entry through a render:satellitepage tag.

- When the code in a CSElement asset is rendered, the code is displayed in the page that called it, and is cached as part of that page (if that page is cached, that is). If you want the output from a CSElement to be cached as a separate pagelet and

have its own cache criteria set for it (timeout value, page criteria values, and so on), your code must invoke that element through a page name. In such a case, you create a SiteEntry asset to accompany your CSElement asset.

When you create and save a SiteEntry asset, you associate a CSElement asset with it. The element in that CSElement asset becomes the root element for the SiteEntry's page entry.

When you save a SiteEntry asset, WebCenter Sites does the following:

- Creates a row in the `SiteEntry` table for the asset.

- Creates a page entry in the `SiteCatalog` table. The root element of the page entry is the element from the CSElement asset that you specified.

- Tracks an approval dependency between the SiteEntry asset and the CSElement asset. Both the SiteEntry asset and its CSElement asset must be approved before the SiteEntry asset can be published.

> **Note:**
>
> Compositional dependencies are also tracked. The SiteEntry defines the page criteria and the default arguments that contain the dependency information. The CSElement records the id of the SiteEntry and CSElement assets into the rendering engine using `render:logdep` tags that are added to the CSElement code stub.

18.3.4 Non-Asset Elements

If you code customizations for the WebCenter Sites interface on the management system, you create elements that are not assets because you do not want them to be published to your delivery system.

For example, when you create workflow elements that implement actions or conditions, you do not create them as CSElement assets. Rather, you use the Oracle WebCenter Sites Explorer tool to manually create an entry in the ElementCatalog table.

Remember that if you create workflow or other custom elements on your delivery system, you must use the CatalogMover utility to copy those elements to the ElementCatalog on your management system.

> **Note:**
>
> You can write code to invoke the mirror engine to mirror your elements. The topic is advanced and beyond the scope of this guide. For assistance, contact Oracle Support at `www.oracle.com/support` or visit `www.oracle.com/accessibility`.

18.4 Creating Template Assets

Template assets render other assets. You can design a template to apply to a specific type of assets or to any assets, irrespective of their types.

- A typed template renders assets of a specific type.

- A typeless template applies to assets of any type. A typeless template is generally used to specify the layout of a page in which assets can then be rendered by the typed templates.

Note:

The only field that makes a template typed or typeless is the For Asset Type field. The purpose of distinguishing templates as typed or typeless is to help developers manage the construction of pages and easily keep track of which templates are responsible for page layout and which for asset rendering.

Before creating a Template asset, complete Before You Begin Creating a Template Asset to determine how you will set template properties (such as the template name) and how you will code the template's element logic. You will then continue to Creating a Template Asset to complete the following steps, using the Admin interface:

- Open the Template Form

- Name and Describe the Template Asset

- Configure the Template's Element

  (To specify its usage, file type, and logic.)

- Configure SiteEntry

  (To specify page and pagelet caching parameters.)

- Configure the Map

  (To support template sharing and site replication.)

- Create a Thumbnail (Optional)

  (To graphically represent the template in its **Inspect** form.)

- Inspect the Template

Information that you enter into the **Template** form will be written to database tables when the template is saved.

Note:

Do not create Template assets directly in the database tables. Doing so will require you to write to several tables and can result in incorrect tracking of dependencies. Instead, use the **Template** form and the procedures in this section to create Template assets. For help with coding the template's element logic (in typed templates), see Coding Elements for Templates and CSElements.

This section includes the following topics:

- Before You Begin Creating a Template Asset

- Creating a Template Asset

18.4.1 Before You Begin Creating a Template Asset

Before you begin creating a Template asset, determine the following:

- *TemplateName* (a name for your Template asset; the value of the **Name** field in the Name screen, Name and Describe the Template Asset).

- Whether the Template asset is to be typed or typeless.

- Whether the Template asset will be shared and whether the site you are working in will be replicated. These considerations determine how you will code the template's element logic.

- Whether to code the Template's element logic in Oracle WebCenter Sites Explorer instead of the Template form. Coding in Oracle WebCenter Sites Explorer, although practiced, is not recommended for the reasons outlined in Using Oracle WebCenter Sites Explorer to Create and Edit Element Logic.

18.4.1.1 Naming a Template Asset

- Once a Template asset is saved, its name cannot be changed.

- WebCenter Sites appends the template name to SiteName (also to AssetTypeName for typed templates). The template name should make sense in relation to SiteName and *AssetTypeName*. WebCenter Sites's naming conventions must not be overridden; names created by WebCenter Sites must not be changed. The table below lists the conventions that use *TemplateName*.

  AssetTypeName is used only for typed templates and represents the value of the template's For Asset Type field in the Name screen (see Name and Describe the Template Asset). *SiteName* is the name of the site to which the template belongs (the site that you are working in as you are creating the template). WebCenter Sites obtains the *SiteName* from the Publication table. (In previous versions of the product, sites were called publications.)

Table 18-2   Naming Conventions Using TemplateName

Template	Naming Conventions	Description
Typed	*AssetTypeName/ TemplateName*	Name of the root element for a typed template.
		This value is written to the **Rootelement** field. This value must not be changed. If the default value is changed, some tags that expect the default value, such as the render:calltemplate tag with the style attribute set to element, will fail.
		Note: The naming convention requires root element names to be unique. You must not have multiple Template assets pointing to the same root element. You can, however, have two SiteEntry assets point to the same element (for example, to specify different default arguments, or different cache criteria depending on the calling scenario).

Table 18-2   (Cont.) Naming Conventions Using TemplateName

Template	Naming Conventions	Description
Typed	*AssetTypeName/ TemplateName*.`xml_or_jsp_ or_html`	Path to the element file of a typed template. This value is written to the `ElementStorage Path/ Filename` when the file type is selected.
Typed	*SiteName/AssetTypeName/ TemplateName*	Name of the page that will be rendered if the template is typed. This value is written to the SiteCatalog Pagename field.
Typeless	*/TemplateName*	Name of the root element for a typeless template. This value is written to the **Rootelement** field. This value must not be changed. If the default value is changed, some tags that expect the default value, such as the `render:calltemplate` tag with the `style` attribute set to `element`, will fail. **Note:** The *AssetTypeName* is omitted, as the template applies to any asset type. The slash is kept to identify the template as typeless. See also the note in the first row of this table.
Typeless	`Typeless/` *TemplateName*`.xml_ or_jsp_or_html`	Path to the element file of a typeless template. This value is written to the **ElementStorage Path/ Filename** when the file type is selected.
Typeless	*SiteName/TemplateName*	Name of the page that will be rendered if the template is typeless. This value is written to the **SiteCatalog Pagename** field. **Note:** The *AssetTypeName/* is omitted, as the template applies to any asset type.

18.4.1.2 Designating a Template as Typed or Typeless

Before creating a Template asset, determine whether it is to be typed or typeless. Once the template is saved, its status as typed or typeless cannot be changed.

18.4.1.3 Template Sharing and Site Replication

Before creating a template, decide how the template and the site you are working in will be used. Your decision determines how you will code the template's element logic (in Configure the Template's Element).

To share your Template asset or make the current site replicable, ensure that the template's element logic does not directly refer to assets, asset types, attribute names, or template names. Instead, you must refer to them indirectly. Use the **Map** screen (Configure the Map) to assign an alias (key) to each value, then hard code the aliases in your template. Use the `render:lookup` tag to retrieve the actual values from the aliases at runtime.

During its execution, the `render:lookup` tag refers to the map to look up the keys and returns the asset-specific information for use in the element logic. This dynamic lookup allows the Template asset (but not the element logic alone) to refer directly to asset data while enabling safe replication and template sharing.

For example, assume a template is named `FSIILayout`, and the site containing this template has a site prefix of `FSII`. If the site is replicated such that the new site's prefix is `New`, and the `FSIILayout` template is copied, then the copy of the template is named `NewLayout`. Referring to the NewLayout template by its hard-coded name (`FSIILayout`) would result in a failure when the template is executed. Instead, the template name is looked up:

```
<%-- Look up the name of the layout template --%>
<render:lookup
        site='<%=ics.GetVar("site")%>'
        varname="LayoutVar"
        key="Layout"
        tid='<%=ics.GetVar("tid")%>'/>

<%-- Look up the name of the wrapper page's site entry.
     Note we want the asset name only, so we must specify
     the match filter. --%>
<render:lookup
        site='<%=ics.GetVar("site")%>'
        varname="WrapperVar"
        key="Wrapper"
        tid='<%=ics.GetVar("tid")%>'
        match=":x"/>
```

To code the element logic, you must have a clear understanding of its design and the map it will refer to. You will have to determine:

- Which keys to create and which name to assign to each key.

- The type of asset information to be looked up:

    - Template Name

    - Asset Type

    - Asset (Type:Name)

    - Asset (Type:ID)

- A value for each key.

- The site to which the map applies.

Additional information about usage of the `render:lookup` tag is given in the *Oracle Fusion Middleware Tag Reference for Oracle WebCenter Sites Reference*.

18.4.2 Creating a Template Asset

You can create a Template asset for a specific asset type or for any asset type.

Note:

Before starting the procedures to create a template asset, read Before You Begin Creating a Template Asset.

To create a template asset, you need to do these tasks:

- Open the Template Form

- Name and Describe the Template Asset

- Configure the Template's Element

- Configure SiteEntry

- Configure the Map

- Create a Thumbnail (Optional)

- Inspect the Template

18.4.2.1 Open the Template Form

1. Open the Admin interface.

2. In the button bar, click **New**.

3. In the list of asset types, select **New Template**.

Note:

For the **New Template** option to be displayed, the Template asset type must be enabled for your site and a **Start Menu** item must be created for it.

---

4. The **Template** form opens.

Note:

If you see a **Choose Assignees** screen instead of the **Template** form, it means that the Template asset you will be creating is associated with a workflow. Select a name (or names) from the Users column and click **Set Assignees**.

Figure 18-1   New Template Name Form

18.4.2.2 Name and Describe the Template Asset

The Name screen is used to identify the template as typed or typeless, assign the template to a category, specify arguments that may be passed to the template, and name keywords by which the template can be located in search routines.

When the Template asset is saved, field values that you specify in the Name screen (with the exception of legal arguments), are written to the Template table, as indicated in the procedures below.

Note:

At any time in the process of creating a template, you can save the template. WebCenter Sites will display the template's Inspect form. To return to the Template form, click the **Edit** link.

In the Name screen, fill in the fields as explained in the following steps:

1. (Required). In the **Name** field, type a descriptive template name that is unique for the template and for the type of asset(s) that the template renders. It is best to choose a name that reflects the function or purpose of the template. See the guidelines in Before You Begin Creating a Template Asset

   Valid Entries

   - Up to 64 alphanumeric characters (the first character must be a letter)

   - Underscores (_)

   - Hyphens (-)

- Spaces (these will be converted to underscores when used in the SiteCatalog pagename for the template)

2. In the **Description** field, type a brief description of the template. You can use up to 128 characters.

3. In the **Source** field, select an option from the drop-down list if your template is derived from a source that you want to note.

4. In the **Category** field, select an option from the drop-down list to place the Template asset into a category.

5. (Required). In the **For Asset Type** field, identify your template as typed or typeless:

   - If you are creating a typeless template (for example to dispatch to typed templates), select **Can apply to various asset types** and skip to step 7.

   - If you are creating a typed template (which renders assets of a certain type), select an asset type. For example, if you are creating a template to render article assets, select **Article** from the drop-down list.

6. (Required for typed templates). In the **Applies to Subtypes** field, select the appropriate subtypes from the menu.

   Note:

   A typed template should be used only for specific subtypes of the asset type that you selected in the preceding field (**For Asset Type**).

   ---

7. In the **Legal Arguments** field:

   a. Enter an argument that may be passed to the template and click **Add Argument**.

   b. In the fields that are displayed:

      - Specify whether the argument is optional or required.

      - Provide a description of the argument (to help you know the purpose of the argument you are creating).

      - Specify legal values (including descriptions) for the argument.

   (You can specify as many arguments and legal values as you require by clicking the **Add Arguments** and **Add Legal Value** buttons.)

8. In the **Keywords** field, enter keywords that you and others can use as search criteria in the Advanced Search form when you search for this template in the future.

9. Click **Continue** to open the Element screen.

> **Note:**
>
> If you chose to save the Template asset, you will notice that WebCenter Sites adds two fields:
>
> - The Status field, which is pre-populated with the editorial status of the Template asset (created, edited, and so on). This field identifies the latest operation that was performed on the Template asset, regardless of whether the Template asset is associated with a workflow.
>
> - The ID field, which is pre-populated with a unique number that WebCenter Sites generates and assigns to the Template asset as its ID. (The ID field corresponds to the tid variable.)

18.4.2.3 Configure the Template's Element

The Element screen is used to create the template's element, define the element file type (XML, JSP, or HTML), provide the element logic, and name the element. For example:

- The **Create Template Element** field offers a choice of XML, JSP, or HTML file types for the element logic, and is used to seed the Element Logic field with standard stub code (which you have to include in any element that you create).

  When you use the **Create Template Element** field to create, for example, a .jsp file, WebCenter Sites adds JSP `taglib` statements and the `RENDER.LOGDEP` tag to the **Element Logic** field by default so that WebCenter Sites can log the compositional dependency between this Template asset and pages that are rendered from this element. For other file types, WebCenter Sites adds code specific to the file type. You will add your own code to the **Element Logic** field.

  See About Dependencies. For help with coding the element logic, see Coding Elements for Templates and CSElements.

- The **Element Storage Path/Filename** field names the file that holds the element logic and specifies the path to the file.

When the Template asset is saved, field values in the Element screen are written to a row (representing the element) in the `ElementCatalog` table.

Note:

About Selecting an Existing Element

In the steps that follow, we assume you are creating a new element for the Template asset. If, however, you are migrating assets from an earlier WebCenter Sites release and want to reuse an existing element, you have to identify the element correctly so that WebCenter Sites can find it and associate it with this Template asset.

To select an existing element, do the following:

1. In the **ElementCatalog Description** field, type a description of the element.

2. In the **Element Storage Path/Filename** field, enter a value according to the naming convention in Table 18-2.

3. In the **Element Parameters** field, specify the variables or arguments that can be passed to the element. See step 7 in Configure the Template's Element.

4. Save and re-open the Template asset.

    WebCenter Sites checks for the existence of the named element:

    If the element has been correctly named, WebCenter Sites recognizes the element and displays its code in the Element Logic field.

    If the named element does not exist (or is incorrectly named), WebCenter Sites does nothing. When you inspect or edit the Template asset, WebCenter Sites displays a message stating that there is no root element in the form. As soon as you code the element and give it the correct name, WebCenter Sites detects it and associates it with the template.

In the Element screen, fill in the fields as explained in this step:

Figure 18-2   Template Element Form

1. In the **Usage** field, specify the intended usage of this template, using this table as a guideline:

Table 18-3   Usage Options

Usage Option	Description
Usage unspecified	Specifies a template that generates HTML. It is unknown whether the template is a Body template (see row 2 of this table) or a url template (see row 3 of this table).
Element is used within an HTML page	Specifies a template that is used inside the `<BODY>...</BODY>` tag of an HTML page. This option characterizes the template as a Body template.
Element is used as a layout	Specifies a layout template that generates a complete HTML page, and is used to render assets in Web Mode.
Element defines a whole HTML page and can be called externally	Specifies a template that generates a complete HTML page and can be used in a url. This option characterizes the template as a url template.
Element is streamed as raw data	Specifies a template that generates raw binary data of an unknown type that is not HTML.

2. In the **Called Templates** field, select the template(s) that this template will call (if they exist).

3. In the **Create Template Element** field, do one of the following:

- To create an .xml file, click **XML**. The code that is pasted in comes from the
  `OpenMarket\Xcelerate\AssetType\Template\modelXML.xml`
  element and can be modified to use custom default logic.

- To create a .jsp file, click **JSP**. The code that is pasted in comes from the
  `OpenMarket\Xcelerate\AssetType\Template\modelJSP.xml`
  element and can be modified to use custom default logic.

- To create an .html file, click **HTML**. The code that is pasted in comes from the
  `OpenMarket\Xcelerate\AssetType\Template\modelHTML.xml`
  element and can be modified to use custom default logic.

WebCenter Sites populates the following fields:

- **Element Logic** field with a header and other auto-generated code.

  For example, if you clicked the **JSP** button, WebCenter Sites enters a tag library directive for each of the WebCenter Sites JSP tag libraries. WebCenter Sites also sets a `RENDER.LOGDEP` (`render:logdep`) tag to mark a compositional dependency between the Template asset and any page or pagelet rendered with the template.

- **Element Storage Path/Filename** field. Do not change the value of this field.

  This field displays the element file name, preceded by the path to the element file. The naming convention is given in Table 18-2.

  When you save the Template asset, the value in the **Element Storage Path/ Filename** field is written to the url column of the `ElementCatalog` table, for the row that represents the element.

4. The **Rootelement** field is pre-populated with the value given in Table 18-2. Do not change the value of this field. If the default value is changed, some tags that expect the default value, such as the `render:calltemplate` tag with the `style` attribute set to `element`, will fail.

5. (Optional). In the **ElementCatalog Description** field, type a description of the element. When you save the Template asset, information in this field is written to the description column for the element entry in the ElementCatalog table.

6. (Required). In the **Element Logic** field, code your element. Be sure to enter all of your code between the two `cs:ftcs` tags.

   If you are using JSP, remove the comments from the `taglib` directives that describe the tag libraries you are using. See Coding Elements for Templates and CSElements.

> **Note:**
>
> **Ensuring Template Sharing or a Replicable Site**
>
> To share your Template asset or make the current site replicable, ensure the template's element logic does not directly refer to assets, asset types, attribute names, or template names. Instead, use the render:lookup tag and prescribed keys as explained in Template Sharing and Site Replication. In Configure the Map you will map the same keys to the asset information that must be accessed for use in the element logic.
>
> **Calling a Template**
>
> Templates should *always* be called by the `render:calltemplate` tag, and never the `render:callelement` tag or `render:satellitepage` tag.

7. The **Element Parameters** field and **Additional Element Parameters** field are used to enter variables or arguments that can be passed to the element, if the site design requires them.

- The **Element Parameters** field corresponds to the resdetails1 column in the ElementCatalog. When you save the template, WebCenter Sites writes the template ID (`tid`) to this field (i.e., to the `resdetails1` column).

- The **Additional Element Parameters** field corresponds to the resdetails2 column in the `ElementCatalog`. WebCenter Sites leaves this field blank.

If your site design requires you to use variables in addition to `tid` in your template element, enter the variables into one of the fields above. Enter them as `name=value` pairs with multiple arguments separated by the ampersand (&) character. For example:

```
MyArgument=value1&YourArgument=value2
```

Each field supports up to 255 characters.

For more information about using variables, see Website Development with Tag Technologies.

8. Click **Continue** to open the SiteEntry screen.

18.4.2.4 Configure SiteEntry

The SiteEntry screen is used to specify caching and pagelet parameters for the page to be rendered by this Template asset.

When the Template asset is saved, field values that you specify in the **SiteEntry** screen are written to the `SiteCatalog` table, as indicated in the procedures below.

In the SiteEntry screen, shown in the following figure, fill in the fields as explained in the steps below:

Figure 18-3   Template Site Entry Form

1. In the **Cache Criteria** field:

    a. WebCenter Sites names the following **reserved** variables as Cache Criteria:

        c,cid,context,p,rendermode,site,sitepfx,ft_ss

    Note:

    The reserved Cache Criteria variables should not be removed. For information about the reserved variables, see Website Development with Tag Technologies.

    ---

    b. If you have to include your own variables as Cache Criteria (for example, **foo**), add them to the existing list. For example:

        c,cid,context,**foo**,p,rendermode,site,sitepfx,ft_ss

> **Note:**
>
> The **Cache Criteria** field names the variables which, in conjunction with SiteCatalog Pagename, define a pagelet as being unique. The variables are used to identify cached pages, which means that the variables are used in the page's cache key.
>
> Only those variables that are specified as Cache Criteria are used by the caching system to create the cache key for cached pages. Therefore, if your site design requires you to use page-level variables in addition to the reserved variables, be sure to designate them as Cache Criteria variables, as shown in this step.

2. If you have to enable the Extra Parameters section in CKEditor, complete these steps:

   a. Move the "Extra Parameters" that were added to the Cache Criteria of the Template to the "Legal Arguments" section of the Template.

   b. When moving the parameters to the Legal Arguments section, it is no longer necessary to prefix the parameters with `fp:`, just use the parameter name.

      For example, instead of using `fp:imageHeight`, just use `imageHeight`.

   c. These parameters are then available in the included template, and can be retrieved in standard ways, such as using `<ics:getvar name="imageHeight"/>`.

   The Extra Parameters section provides a way of passing custom parameters, such as image dimensions, to the template. These extra parameters will be available in the **Include asset link** and **Add asset link** dialog boxes. The parameter names (`imageHeight` and `imageWidth` in our example) will be displayed in CKEditor's Extra Parameters section, as options in the **Name** menu. The Value field enables the user to specify a value for the chosen parameter.

   When the Template asset is saved, the Cache Criteria variables are written to the `pagecriteria` column in the `SiteCatalog` table.

3. (Optional). The **Access Control Lists** field corresponds to the `acl` column in the `SiteCatalog` table. To allow only certain visitors to request this page, select the ACLs that the visitors must have to see the page.

4. The **Rootelement** field is pre-populated with a value that is shown in .

5. The **Cache Rules** field corresponds to the `cscacheinfo` and `sscacheinfo` columns in the `SiteCatalog` table. Do one of the following:

   - Select **Cached** if the pagelet to be rendered by this template's element must be cached. The pagelet is set to be cached forever. The cache will be flushed by CacheManager's active cache management logic. This option sets both WebCenter Sites and Satellite Server caching conditions.

   - Select **Uncached** to turn off caching for the pagelet to be rendered by this template's element. This option sets both WebCenter Sites and Satellite Server caching conditions.

- Select **Advanced** to set caching rules individually for WebCenter Sites and Satellite Server. Selecting **Advanced** displays two additional fields: one for WebCenter Sites caching and one for Satellite Server caching.

Note:

CacheManager is designed to manage the lifecycle for cached pages on both WebCenter Sites and Satellite Server. It is designed to operate with pages that are set to be cached forever. If the cache expires on WebCenter Sites before it expires on Satellite Server, CacheManager will fail to flush the cache properly and invalid pages may be served from cache. Only advanced users should configure these settings manually.

For more information about page caching settings, see Understanding Page Design and Caching.

---

1. **SiteCatalog Pagename** field. Do not change the value of this field. This field is pre-populated with the name of the page entry. The page naming convention is given in Table 18-2.

2. In the **Pagelet parameters** section, you can enter pagelet parameters (name-value pairs), which will be passed into the template each time it is executed. The **Pagelet parameters** section supports a total of 510 characters.

    Note:

    The **Pagelet parameters** section is pre-populated with the following default pagelet parameters (reserved variables that were named in step 1, including their values:

    ```
    site, sitepfx, rendermode
    ```

    The default parameter values will be overwritten if they are explicitly specified when the template is called.

    -   If you are specifying a pagelet parameter in this step, make sure to list its name as a Cache Criteria variable (see step 1).

    -   If you named your own Cache Criteria variables (in step 1), the variables are listed in the **Page parameters** section. If you do not specify values for these parameters, WebCenter Sites ignores the parameters.

    -   If you want to control the allowed HTTP methods for this page entry, add a pagelet parameter with the name `methods` and a value with comma-separated HTTP methods. For example, `methods=GET,OPTIONS`.

    When the Template asset is saved, the name-value pairs that are specified as **Pagelet parameters** are written to either the `resargs1` or `resargs2` column of the `SiteCatalog` table. The column to which they are written is not important and is managed automatically. (Each column supports up to 255 characters.)

3. Click **Continue** to open the Thumbnails screen.

4. Click **Continue** to open the Map screen.

> **Note:**
>
> You will return to the **Thumbnail** screen after you have completed creating the Template asset and saved the Template asset.

18.4.2.5 Configure the Map

The purpose of mapping is to enable site replication and the sharing of Template assets.

> **Note:**
>
> Skip this step if you are designing a non-replicable site.

Using the Map form, you will:

- Map each key in the `render:lookup` tags of the template's element logic to a value that will be used by the element logic.

- Map each key's value to the asset information that must be used in the element logic: asset, asset type, attribute name, or template name.

When the Template asset is saved, the map is written to the `Template_Map` table.

In the Map form, fill in the fields as explained in this step:

1. The **Key** field represents a value that the element logic will look up. Enter the key that is named in a `render:lookup` tag of the element logic.

2. The **Type** field identifies the type of asset information to be accessed. Select one of the following options:

   - **Template Name**: Maps a template name to the key value (which you will specify in the **Value** field, in the next step). The information that will be accessed is a template name that matches the value that you will specify in the next step. For an example, see the next figure.

   - **Asset Type**: Maps an asset type to the key value. The information that will be accessed is an asset type, equal to the value that you will specify in the next step.

   - **Asset (Type:Name)**: Maps an attribute type:name to the key value. The information that will be accessed is an asset whose type and name match the value that you will specify in the next step.

   - **Asset (Type:ID)**: Maps an attribute type:ID to the key value. The information that will be accessed is an asset whose type and name match the value that you will specify in the next step.

Figure 18-4   Template Asset: Sample Map

3. In the **Value** field, enter a value for the key. This value will be looked up by the element logic when the Template is executed.

4. In the **siteid** field, select the name of the site to which the mapping applies.

5. To add a key, click **Add Another** and repeat the steps in this section.

6. When you have completed creating your template, save the template (click **Save Changes**).

    WebCenter Sites displays the template's Inspect form.

18.4.2.6 Create a Thumbnail (Optional)

A thumbnail graphically assists template users in determining how your Template asset lays out pages or renders content. The thumbnail that you create will be displayed in the Template's Inspect form.

When the Template asset is saved, the name of the thumbnail file is written to the urlthumbnail column of the Template_Thumb table.

To create a thumbnail, complete the following steps:

1. Preview your Template asset.

2. Capture the preview as an image file and save it to a file system.

3. Open the Template form and click **Thumbnail** at the top of the screen.

    The Template Thumbnail form opens (see the next figure).

4. In the **Thumbnail Image** field, enter (or browse for) the path to the image file that you created in step 2.

Figure 18-5   Template Thumbnail Form

5. To display the thumbnail in the Inspect form:

   a. Save the template (click the **Save** icon).

      WebCenter Sites uploads the image file to the WebCenter Sites database and displays template's Inspect form.

   b. In the Inspect form, scroll down to the Thumbnail Image section. If the displayed image is too large or too small, resize the image in its source file and repeat steps 4 and 5.

6. To operate in the image in the Thumbnail screen:

   a. Scroll to the top of the Inspect form, and click the **Edit** link.

   b. At the top of the Template form, click **Thumbnail**.

7. To copy, send, and perform other operations on the thumbnail, right-click the thumbnail and select an option.

8. To delete the thumbnail, select **Delete thumbnail image?** and click **Save Changes**.

   WebCenter Sites displays the template Inspect form.

18.4.2.7 Inspect the Template

When you have finished creating the Template asset and clicked **Save**, WebCenter Sites does the following:

- Writes to the database tables:

  - Creates a template entry in the Template table.

  - Creates an element entry in the `ElementCatalog` table, using the `AssetTypeName/TemplateName` naming convention. If the element was coded in the template form (rather than Oracle WebCenter Sites Explorer), WebCenter Sites also creates the element file.

  - Determines the name of the site that the template belongs to and creates a page entry in the `SiteCatalog` table using the `SiteName/AssetTypeName/TemplateName` naming convention.

  - Sets the name of the root element of the new `SiteCatalog` page entry to the name of the `ElementCatalog` entry.

  - Creates a thumbnail entry in the `Template_Thumb` table.

- – Creates a map entry in the Template_Map table.

- Displays the Inspect form, which provides the following kinds of information:

  - – Information in the Name screen (standard summary information, such as asset name, description, status, source, and ID, for assets of all types).

  - – Information in the Element screen (root element, element logic, path to the element file, and tid).

  - – Information in the SiteEntry screen (SiteCatalog pagename, pagelet parameters, cache criteria, and the ACLs of users who are authorized to view the page).

  - – Information in the Thumbnail screen (a thumbnail image, if one was chosen).

  - – Information in the Map screen (a map of key-value-asset information, if the site was designed to be replicable, or the template is sharable).

  - – If you have shared the Template asset, the Inspect form also lists all of the additional page entries in the SiteCatalog for this Template asset (there is a page entry for each site that the template is shared with).

Figure 18-6   Inspect Form Showing Element Parameters
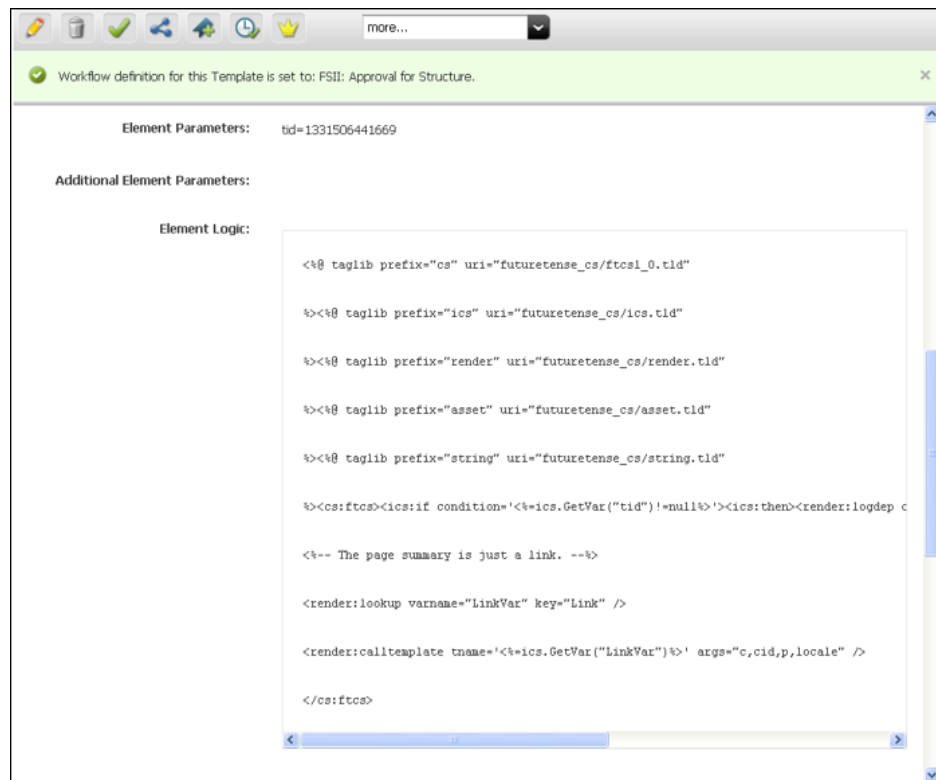
Figure 18-7 Inspect Form Showing Site Entries and Maps
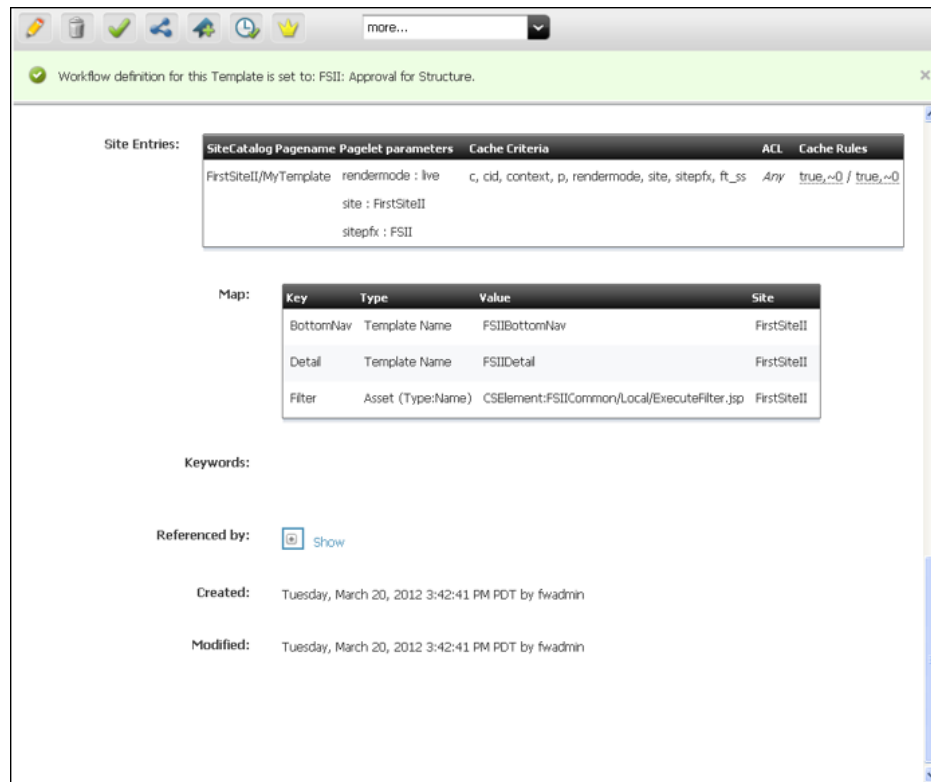
18.5 Creating CSElement Assets

When you create a CSElement asset, you do three things: you create an asset, you code an element for the asset, and you configure a key-value-asset information map (similar to the map for a Template asset).

To create a CSElement asset, you must first complete Before You Begin Creating a CSElement to determine how you will set CSElement properties that cannot (or must not) be changed once the CSElement is saved, and how you will code the CSElement's element logic. You will then continue to Creating a Template Asset to complete the following steps, using the Admin interface:

- Open the CSElement Form

- Name and Describe the CSElement Asset

- Configure the Element

  (To specify its file type and logic.)

- Configure the Map

  (To support CSElement sharing and site replication.)

- Save and Inspect the CSElement

- Add the CSElement to Bookmarks

  (If you plan to use the CSElement as a root element for a Site Entry asset. See Creating SiteEntry Assets.)

Information that you enter into the **CSElement** form will be written to database tables when the CSElement asset is saved, as indicated in the procedures below.

Note:

Do not create CSElement assets directly in the database tables. Doing so will require you to write to several tables and can result in incorrect tracking of dependencies. Instead, use the **CSElement** form and the procedures in this section to create CSElement assets. For help with coding the CSElement logic, see Coding Elements for Templates and CSElements.

Topics:

- Before You Begin Creating a CSElement
- Creating a SiteEntry Asset

18.5.1 Before You Begin Creating a CSElement

Before you begin creating a CSElement asset, you must determine several things:

- A name for your CSElement asset.

- Whether your CSElement will be sharable and the site replicable. These considerations determine how you will code the CSElement's element logic.

- Whether you plan to code the CSElement's element logic in Oracle WebCenter Sites Explorer instead of the CSElement form. This approach is not recommended for the reasons outlined in Using Oracle WebCenter Sites Explorer to Create and Edit Element Logic.

18.5.1.1 Naming the CSElement

- Once the CSElement asset is saved, its name cannot be changed.

- The CSElement logic file takes the name of the CSElement (followed by the file extension:

  CSElementName.xml_or_jsp_or_html

  The name of the CSElement logic file must not be changed.

18.5.1.2 CSElement Sharing and Site Replication

Before creating a CSElement, decide whether the CSElement must be shared or the site you are working in must be replicable. If so, the CSElement logic will be coded in the same way. If sharing and replication are not required, you will skip key-value mapping (see Configure the Map).

For information about coding element logic to support CSElement sharing and site replication, see Template Sharing and Site Replication. The information applies without exception to CSElement assets.

18.5.2 Creating a CSElement Asset

You can create a CSElement asset through the WebCenter Sites interface.

> **Note:**
>
> Before starting the procedures in this section, read Before You Begin Creating a CSElement for information about creating CSElement assets.

- Open the CSElement Form

- Name and Describe the CSElement Asset

- Configure the Element

- Configure the Map

- Save and Inspect the CSElement

- Add the CSElement to Bookmarks

18.5.2.1 Open the CSElement Form

1. Open the Admin interface.

2. In the button bar, click **New**.

3. In the list of asset types, select **New CSElement**.

The CSElement form opens.

> **Note:**
>
> For the New CSElement option to be displayed, the CSElement asset type must be enabled for your site and a Start Menu item must be created for it.

4. Continue with Name and Describe the Template Asset.

> **Note:**
>
> If you see a Choose Assignees screen instead of the CSElement form, it means that the CSElement you will be creating is associated with a workflow. Select a name (or names) from the Users column and click **Set Assignees**. Continue with Name and Describe the Template Asset.

18.5.2.2 Name and Describe the CSElement Asset

The Name screen is used to define metadata about the CSElement. From this metadata, a developer can identify what the CSElement does and the arguments it uses to perform its function.

> **Note:**
>
> At any time in the process of creating a CSElement, you can save the CSElement. WebCenter Sites will display the CSElement's Inspect form. To return to the CSElement form, click the **Edit** link.

To name and describe the CSElement, in the Name screen, fill in the fields as explained in the following steps:

Figure 18-8   CSElement Name Form

1. (Required). In the **Name** field, type a unique, descriptive name for the CSElement asset. It's best to use a name that describes what the CSElement does.

    Valid Entries

    - Up to 64 alphanumeric characters (the first character must be a letter)

    - Underscores (_)

    - Hyphens (-)

    - Spaces (these will be converted to underscores when used in the SiteCatalog pagename for the template)

    Note:

    Make sure you have chosen a name for your CSElement asset using the guidelines inBefore You Begin Creating SiteEntry Assets.

    ---

2. In the **Description** field, type a brief description of the CSElement asset. You can enter up to 128 characters.

3. In the **Legal Arguments** field:

    a. Enter an argument that may be passed to the CSElement and click **Add Argument.**

    b. In the fields that are displayed:

        - Specify whether the argument is optional or required.

        - Provide a description of the argument (to help you know the purpose of the argument you are creating).

        - Specify legal values (including descriptions) for the argument.

    (You can specify as many arguments and legal values as you require by clicking the **Add Arguments** and **Add Legal Value** buttons.)

4. Click **Continue** to open the Element screen.

18.5.2.3 Configure the Element

The Element form (Figure 18-9) is used to create the CSElement's element, define the element file type (XML, JSP, or HTML), provide the element logic, and name the element. For example:

- The **Create Element** field offers a choice of XML, JSP, or HTML file types for the element logic, and is used to seed the Element Logic field with standard stub code (which you have to include in any element that you create).

- When you use the **Create Element** field to create, for example, a .jsp file, WebCenter Sites adds JSP `taglib` statements and the `render.logdep` tag to the **Element Logic** field by default so that the compositional dependency between this CSElement asset and pages that are rendered from this element is logged. For other file types, WebCenter Sites adds code specific to the file type. You will add your own code to the **Element Logic** field.

  For information about dependencies, see About Dependencies. For help with coding the element logic, see Coding Elements for Templates and CSElements.

- The **Element Storage Path/Filename** field names the file that holds the element logic and specifies the path to the file.

When the CSElement is saved, field values in the Element screen are written to a row (representing the element) in the `ElementCatalog` table.

Note:

About Selecting an Existing Element

In the steps that follow, we assume you are creating a new element for the CSElement asset. If, however, you are migrating assets from an earlier WebCenter Sites release and want to reuse an existing element, you have to identify the element correctly so that WebCenter Sites can find it and associate it with the CSElement asset.

To select an existing element

- (Optional). In the **ElementCatalog Description** field, type a description of the element.

- In the **Element Storage Path/Filename** field, enter a value according to the convention in Naming the CSElement.

- If your site design requires it, enter the appropriate arguments in the element parameter fields. For instructions, see step 5.

- Save and re-open the CSElement asset.

WebCenter Sites checks for the presence of an element with the correct name:

If the element has been correctly named, WebCenter Sites recognizes the element and displays its code in the Element Logic field.

If the named element does not exist (or is incorrectly named), WebCenter Sites does nothing. When you inspect or edit the CSElement asset, WebCenter Sites displays a message stating that there is no root element in the form. As soon as you code the element and give it the correct name, WebCenter Sites detects it and associates it with the CSElement asset.

To configure a new element, in the Element form, fill in the fields as explained in the steps below:

Figure 18-9   CSElement Element Form

1. In the **Create Element** field, do one of the following:

   - To create an .xml file, click **XML**. The code that is pasted in comes from the `OpenMarket\Xcelerate\AssetType\CSElement\modelXML.xml` element and can be modified to use custom default logic.

   - To create a .jsp file, click **JSP**. The code that is pasted in comes from the `OpenMarket\Xcelerate\AssetType\CSElement\modelJSP.xml` element and can be modified to use custom default logic.

   - To create an .html file, click **HTML**. The code that is pasted in comes from the `OpenMarket\Xcelerate\AssetType\CSElement\modelHTML.xml` element and can be modified to use custom default logic.

   WebCenter Sites populates the following fields:

   - **Element Storage Path/Filename** field. Do not change the value of this field.

     This field displays the element file name preceded by the path to the element file. By default, the file takes the name of the CSElement asset (entered in step 1) followed by the file extension:

     `CSElementName.xml_or_jsp_or_html`

     When you save the CSElement asset, the value in this field is written to the url column of the ElementCatalog table, for the row that represents the element.

   - **Element Logic** field with a header and other information.

     For example, if you clicked the **JSP** button, WebCenter Sites sets a tag library directive for some common WebCenter Sites JSP tag libraries (`asset`, `siteplan`, `render`). WebCenter Sites also sets the beginning and ending cs:ftcs tags, and a `RENDER.LOGDEP` (`render:logdep`) tag to mark a compositional dependency between the CSElement asset and any page or pagelet rendered by the element.

2. The **Rootelement** field is pre-populated with the name of the element file *(CSElementName.xml_or_jsp_or_html)*. Do not change the value of this field.

   The root element is listed by this name in the `ElementCatalog` table. When you create code that calls this element (`RENDER.CALLELEMENT`), this is the name you should use. It uses the name of the CSElement asset by default.

3. (Optional). In the **ElementCatalogDescription** field, type a description of the element.

   When you save the CSElement asset, information in this field is written to the `description` column for the element entry in the ElementCatalog table.

4. (Required). In the **Element Logic** field, code your element. Be sure to enter all of your code before the ending `cs:ftcs` tag.

   If you are using JSP, remove the comments from the `taglib` directives that describe the tag families you are using.

   For help with this step, see Coding Elements for Templates and CSElements.

Note:

Ensuring Template Sharing or a Replicable Site: To share your CSElement or make the current site replicable, ensure the CSElement's element logic does not directly refer to assets, asset types, attribute names, or template names. Instead, use the render:lookup tag and prescribed keys as explained in Template Sharing and Site Replication. In Configure the Map you will map the keys to the asset information that must be accessed for use in the element logic.

Calling a Template: Templates should *always* be called by the `render:calltemplate` tag, and never the `render:callelement` tag or `render:satellitepage` tag.

5. (Optional). The **Element Parameters** field and **Additional Element Parameters** fields are used to enter variables or arguments that can be passed to the element, if the site design requires them.

- **Element parameters** field. WebCenter Sites populates this field with the CSElement ID (`eid`), generated by WebCenter Sites as a unique identifier of the CSElement asset. **Do not change or delete this value**.

  This field corresponds to the resdetails1 column of the ElementCatalog table. When you save the CSElement, WebCenter Sites writes the CSElement ID to the `resdetails1` column, in the row that represents the CSElement.

- **Additional Element Parameters** field. WebCenter Sites leaves this field blank.

  This field corresponds to the `resdetails2` column of the `ElementCatalog`.

If your site design requires you to use variables in addition to eid, enter the variables into one of the fields above. Enter them as *name=value* pairs with multiple arguments separated by the ampersand (&) character. For example:

```
MyArgument=value1&YourArgument=value2
```

Each field supports up to 255 characters.

For more information about WebCenter Sites variables, including scope and precedence, see Website Development with Tag Technologies.

6. Click **Continue** to open the Map screen.

18.5.2.4 Configure the Map

The purpose of mapping is to enable site replication and sharing of CSElement assets. The concepts behind mapping are identical to those for Template assets. They are explained in Template Sharing and Site Replication.

Note:

Skip this section if you are designing a non-replicable site or a CSElement asset that is not shared.

Using the Map form, you will:

- Map each key in the `render:lookup` tag of the element logic to the value that must be used in the element logic.

- Map each key's value to the asset information that must be used in the element logic: asset, asset type, attribute name, or template name.

When the CSElement asset is saved, the map is written to the `CSElement_Map` table.

In the Map form, fill in the fields as explained in this step:

Figure 18-10   CSElement Map Form

1. The **Key** field represents the value that the element logic will look up. In this field, enter the key that is named in a render:lookup tag of the element logic.

2. The **Type** field identifies the type of asset information to be accessed. Select one of the following options:

   - **Template Name**: Maps a template name to the key value (which you will specify in the Value field, in the next step). The information that will be accessed is a template name that matches the value you will specify in the next step. (For an example, see the next figure.)

   - **Asset Type**: Maps an asset type to the key value. The information that will be accessed is an asset type, equal to the value that you will specify in the next step.

   - **Asset (Type:Name)**: Maps an attribute type:name to the key value. The information that will be accessed is an asset whose type and name match the value that you will specify in the next step.

   - **Asset (Type:ID)**: Maps an attribute type:ID to the key value. The information that will be accessed is an asset whose type and name match the value that you will specify in the next step.

Figure 18-11   CSElement Asset: Sample Map

3. In the **Value** field, enter a value for the key. This value will be looked up by the element logic when the CSElement asset is invoked.

4. In the **siteid** field, select the name of the site to which the mapping applies.

5. To add a key, click **Add Another** and repeat the steps in this section.

18.5.2.5 Save and Inspect the CSElement

When you have finished creating the CSElement asset, click **Save**.

- WebCenter Sites writes to the database tables.

  - Creates a row in the `CSElement` table for the CSElement asset, where it enters the CSElement name and description that you specified in the previous steps.

  - Creates an element entry in the `ElementCatalog` table using values specified in the Element screen:

    * The value of the **Rootelement** field is used to position the element file in the appropriate folder.

    * The value of the **Element Storage Path/Filename** field is written to the url column.

    * The value of the `eid` variable is set to the ID of the CSElement asset in the `resdetails1` column.

- Flushes the pagecache of any pagelets that call this element.

- Displays the Inspect form (Figure 18-12), which provides the following kinds of information:

- Information in the Name screen (standard summary information, such as asset name, description, status, and ID, for assets of all types).

- Information in the Element screen (root element, element logic, path to the element file, and the element's eid).

- Information in the Map screen (a map of key-value-asset information, if the site was designed to be replicable, or the template is sharable).

- **Preview with Arguments** button, enabling you to preview the page(s) rendered by the SiteEntry asset.

18.5.2.6 Add the CSElement to Bookmarks

Note:

Complete the steps in this section if you are planning to use your CSElement to create a SiteEntry asset. This step makes the CSElement available for selection in WebCenter Sites's tree by adding it to your Bookmarks.

If you are not planning to create the SiteEntry asset in this session, you want to add the CSElement to your Bookmarks so that you can easily find it later.

To add the CSElement to Bookmarks, do the following:

1. Run a search on the CSElement asset you created.

2. In the results list, select the check box in the right-hand column to add the CSElement to **Bookmarks**.

   This CSElement is listed in WebCenter Sites's tree, in the **Bookmarks** tab, where it is a selectable option for SiteEntry assets.

3. Create the SiteEntry asset. See Creating SiteEntry Assets.

Figure 18-12   CSElement Asset: Sample Inspect Form

18.6 Creating SiteEntry Assets

When you create a SiteEntry asset, you are creating both an asset and a page entry in the SiteCatalog table. The fields in the first part of the SiteEntry form define the page entry as an asset. The rest of the fields provide information about the page entry as a WebCenter Sites page, information that is written to the SiteCatalog table.

To create a SiteEntry asset, you must first complete Before You Begin Creating a CSElement to create its root element and determine how you will set SiteEntry properties (such as SiteEntry name). You will then continue to Creating a SiteEntry Asset to complete the following steps, using the Admin interface:

- Open the SiteEntry Form

- Create the SiteEntry Asset

- Save and Inspect the SiteEntry Asset

Information that you enter into the **SiteEntry** form will be written to database tables when the CSElement asset is saved, as indicated in the procedures below.

> **Note:**
>
> Do not create SiteEntry assets directly in the database tables. Doing so will require you to write to several tables and can result in incorrect tracking of dependencies. Instead, use the **SiteEntry** form and the procedures in this section to create SiteEntry assets.

This section includes the following topics:

- Before You Begin Creating SiteEntry Assets

- Creating a SiteEntry Asset

18.6.1 Before You Begin Creating SiteEntry Assets

Before you begin creating a SiteEntry asset, complete the following steps:

1. Create a root element for the page entry:

    a. Create a CSElement asset. For instructions, see Creating CSElement Assets.

    (A root element is required for any page entry. The root element is the element of the CSElement, which you will select for the SiteEntry asset.)

    b. Make the CSElement available. For instructions, see Add the CSElement to Bookmarks.

    (To specify a CSElement for your SiteEntry asset, you will select the CSElement from the **Bookmarks** tab in WebCenter Sites's tree, or the **History** tab if you created the CSElement in the current session).

2. Determine a name for your SiteEntry asset. The same name will be assigned to the page. Neither the SiteEntry name nor the page name can be changed once the SiteEntry asset is saved.

18.6.2 Creating a SiteEntry Asset

You can create a SiteEntry asset through the WebCenter Sites interface.

To create a SiteEntry asset, do these tasks:

- Open the SiteEntry Form
- Create the SiteEntry Asset
- Save and Inspect the SiteEntry Asset

> **Note:**
>
> Before starting the procedures in this section, read Before You Begin Creating SiteEntry Assets for information about creating SiteEntry assets.

18.6.2.1 Open the SiteEntry Form

1. Open the Admin interface.

2. In the button bar, click **New**.

3. In the list of asset types, select **New SiteEntry**.

    The SiteEntry form opens.

> **Note:**
>
> For the **New SiteEntry** option to be displayed, the SiteEntry asset type must be enabled for your site and there must be a **Start Menu** item created for it.

4. If you see a **Choose Assignees** screen instead of the **SiteEntry** form, it means that the SiteEntry you will be creating is associated with a workflow. Select a name (or names) from the Users column and click **Set Assignees**.

18.6.2.2 Create the SiteEntry Asset

In the SiteEntry form, fill in the fields as explained in this step:

1. (Required). In the **Name** field, type a descriptive name for the SiteEntry asset. It's best to use a name that describes the purpose of the page.

   Valid Entries

   - Up to 64 alphanumeric characters (the first character must be a letter)

   - Underscores (_)

   - Hyphens (-)

   - Spaces (these will be converted to underscores when used in the `SiteCatalog` pagename for the template).

2. In the **Description** field, type a brief description of the SiteEntry asset. You can enter up to 128 characters.

3. (Required). Click in the **Pagename** field to automatically populate it with the name of the page entry and the path to the page entry (for example: `FSIICommon/SideNav/ProductView`). Do not change the value of this field.

   > **Note:**
   >
   > The value in this field is the name of the page entry (which will be stored in the `SiteCatalog` table when the SiteEntry is saved). When you create code that calls this SiteEntry asset (`RENDER.SATELLITEPAGE`), this is the name you should use.

4. To use the row of a pre-existing page entry in the `SiteCatalog` table, select the **Map to existing SiteCatalog entry** option.

5. (Required). In the **Rootelement** field, select the appropriate CSElement asset from the tree and click **Add Selected Items**.

   > **Note:**
   >
   > Only one CSElement can be added.

6. In the **Wrapper page** field, select one of the options to specify whether the asset you are creating is a wrapper page. Selecting the **NO** option displays the Pagelet only field.

7. If the **Pagelet only** field is displayed, select one of the options to specify whether the asset you are creating is a pagelet.

8. In the **Pagelet parameters** section, you can enter pagelet parameters (name-value pairs), which will be passed into the page or pagelet each time it is executed. The Pagelet parameters section supports a total of 510 characters.

> **Note:**
>
> Keep in mind the following:
>
> - The **Pagelet parameters** section is pre-populated with the following default parameters (reserved variables that are named by default in the Cache Criteria field (next step), including their values: `site, seid, sitepfx, rendermode`
>
>   The default values will be overwritten if they are explicitly specified when the page or pagelet is called.
>
> - If you are specifying a pagelet parameter in this step, make sure to list its name as a Cache Criteria variable in the next step.
>
> - If you want to control the allowed HTTP methods for this page entry, add a pagelet parameter with the name `methods` and a value with comma-separated HTTP methods. For example, `methods=GET,OPTIONS`.

When the SiteEntry asset is saved, the name-value pairs that are specified as **Pagelet parameters** are written to either the `resargs1` or `resargs2` column of the `SiteCatalog` table. The column to which they are written is not important and is managed automatically. (Each column supports up to 255 characters.)

9. In the **Cache Criteria** field:

   a. WebCenter Sites names the following **reserved** variables as Cache Criteria:

   ```
   rendermode,seid,site,sitepfx,ft_ss
   ```

   > **Note:**
   >
   > The reserved Cache Criteria variables should not be removed.

10. If you have to include your own variables as Cache Criteria (for example, foo), add them to the existing list. For example

    foo,rendermode,seid,site,sitepfx,ft_ss

    > **Note:**
    >
    > The Cache Criteria field names the variables which, in conjunction with Pagename, define a pagelet as being unique. The variables are used to identify cached pages, which means that the variables are used in the page's cache key.
    >
    > Only those variables that are specified as Cache Criteria are used by the caching system to create the cache key for cached pages. Therefore, if your site design requires you to use page-level variables in addition to the reserved variables, be sure to designate them as Cache Criteria variables, as shown in this step.

When the SiteEntry asset is saved, Cache Criteria variables and their values are written to the `pagecriteria` column in the `SiteCatalog` table.

11. The **Cache Rules** field corresponds to the cscacheinfo and sscacheinfo columns in the SiteCatalog table. Do one of the following:

- Select **Cached** if the pagelet to be rendered by this SiteEntry's CSElement must be cached. The pagelet is set to be cached forever. The cache will be flushed by CacheManager's active cache management logic. This option sets both WebCenter Sites and Satellite Server caching conditions.

- Select **Uncached** to turn off caching for the pagelet to be rendered by this SiteEntry's CSElement. This option sets both WebCenter Sites and Satellite Server caching conditions.

- Select **Advanced** to set caching rules individually for WebCenter Sites and Satellite Server. Selecting **Advanced** displays two additional fields: one for WebCenter Sites caching and one for Satellite Server caching.

Note:

CacheManager is designed to manage the lifecycle for cached pages on both WebCenter Sites and Satellite Server. It is designed to operate with pages that are set to be cached forever. If the cache expires on WebCenter Sites before it expires on Satellite Server, CacheManager will fail to flush the cache properly and invalid pages may be served from cache. Only advanced users should configure these settings manually.

---

12. The **Access Control Lists** field corresponds to the acl column in the SiteCatalog table. to allow only certain visitors to request this page, enter the ACLs that visitors must have to see the page.

18.6.2.3 Save and Inspect the SiteEntry Asset

When you have finished creating the SiteEntry asset, click **Save**. WebCenter Sites does the following:

- Writes to the database tables:

  – Creates a row in the SiteCatalog table for the SiteEntry asset, where it enters the values that you specified in the previous steps.

- Displays the Inspect form, which provides the following information:

  – Standard summary information (asset name, description, status, ID) and the page entry criteria you specified in the previous steps.

  – **Preview with Arguments** button, enabling you to preview the page(s) rendered by the SiteEntry asset.

18.7 Managing Template, CSElement, and SiteEntry Assets

You can preview, edit, and delete Template, CSElement, and SiteEntry assets. If you're developing multiple sites, sharing these assets with other sites can save development time.

See these topics:

- Designating Default Approval Templates (Static Publishing Only)

- Editing Template, CSElement, and SiteEntry Assets

- Sharing Template, CSElement, and SiteEntry Assets

- Deleting Template, CSElement, and SiteEntry Assets

- Previewing Template, CSElement, and SiteEntry Assets

18.7.1 Designating Default Approval Templates (Static Publishing Only)

When assets are approved for a publishing destination that uses the Export to Disk publishing method, the approval system examines the template assigned to the asset to determine its dependencies.

If you design your online site to render assets with multiple templates (a text-only version and a summary version and a full version for the same type of asset, for example), you should create a template that contains a representative set of approval dependencies for all of the templates, and then specify that template as the Default Approval Template for the asset type.

See Approval Templates for Export to Disk.

To designate that a template is the default approval template:

1. In the **General Admin** tree, expand the **Admin** node, then select **Publishing**, then **Destinations**, and then **Static**.

2. Under the name of a static destination, select **Set Default Templates**.

3. In the Default Templates form, click **Edit**.

4. In the edit form, select a default template for each asset type. If you are using the Subtype feature for any of your asset types, you can designate a default approval template for each subtype of that asset type.

5. When you have finished, click **Save**.

18.7.2 Editing Template, CSElement, and SiteEntry Assets

Creating a `Template`, `CSElement`, or `SiteEntry` asset also creates an entry in the SiteCatalog table or ElementCatalog table or both. The names of those entries are based on the asset's name, and for Template assets, the asset type and the site the template belongs to. Because these naming dependencies exist, the following restrictions apply when you edit a `Template`, `CSElement`, or `SiteEntry` asset:

- You cannot rename a `Template`, `CSElement`, or `SiteEntry` asset after it has been saved.

- For `Template` assets, you cannot change the asset type selected in the **Asset Type** field after the Template asset has been saved.

- For a `Template` or `CSElement` asset, you cannot change the name of the root element.

- For a `SiteEntry` asset, you cannot change the name of the page entry.

> **Caution:**
>
> If you have manually created one or more site entries that point to a `Template` (using Oracle WebCenter Sites Explorer) and then edit the `Template` through the **Admin** interface, the manually created site entries are automatically deleted.

For the basic procedure on editing assets, see Managing Access to Asset Types Using Start Menus in *Oracle Fusion Middleware Administering Oracle WebCenter Sites*.

18.7.3 Sharing Template, CSElement, and SiteEntry Assets

When you share a CSElement, template, or SiteEntry asset, WebCenter Sites creates a row in the `AssetPublication` table for each site that you share the asset with.

Additionally, for **Template assets only**, WebCenter Sites does the following:

- Creates a new SiteCatalog page entry for each site that you share the asset with. It uses the name of the site in the name of the page entry. All of the new page entries point to the same root element, the template element.

  > **Note:**
  >
  > Do not change the root elements of these page entries. All page entries for a shared template must point to the same root element.

- Lists all the other page entries for the shared template that share this root element in the Inspect form.

For the basic procedure on sharing assets, see Sharing Blog Assets in *Oracle Fusion Middleware Administering Oracle WebCenter Sites*.

> **Note:**
>
> For templates and CSElements to be sharable, their element logic must not be hard-coded with asset type names, attribute names, template names, or IDs. Instead, use the `render:lookup` tag and hard-code the keys for which you have created a map that the `render:lookup` tag can refer to look up asset information for use in the element logic.

18.7.4 Deleting Template, CSElement, and SiteEntry Assets

WebCenter Sites does not allow you to delete an asset if there is another asset using it. However, it does not check to see whether a template or CSElement is referenced by the code in other template or CSElement elements.

Before you delete a template or SiteEntry asset, be sure to remove any page calls to that asset's page entry from your elements. Before you delete a CSElement asset, be sure to remove any element calls to that asset's root element from your other elements.

When you delete an asset, WebCenter Sites does the following:

- Changes the value of the asset's name column in the Template, CSElement, or SiteEntry table (depending on the asset type) to its object ID.

- Changes the value of the asset's `status` column in the Template table to VO, for void.

- For templates, deletes all the SiteCatalog table entries (if the template is shared, there are as many page entries as there are sites that the template is shared with) and the ElementCatalog table entry for the template.

- For CSElements, deletes the `ElementCatalog` table entry for the asset.

See Deleting Index Data in *Oracle Fusion Middleware Administering Oracle WebCenter Sites*.

18.7.5 Previewing Template, CSElement, and SiteEntry Assets

Because template, CSElement, and SiteEntry assets provide logic and code for formatting other assets, you preview assets of these types differently from the way you preview your content assets.

18.7.5.1 Templates and Preview

You preview a template by previewing an asset and selecting the template that you want to use to render the asset. WebCenter Sites invokes the code in the template and renders a page with the asset as the content.

18.7.5.2 CSElement and SiteEntry Assets and Preview

You preview `CSElement` and `SiteEntry` assets directly. If the element that will be called has self-contained context, a banner that does not expect variables or arguments. For example, you can simply click the **Preview** icon. But when the results of the rendered element depend on values that are passed to it, you must manually set those values in the `CSElement` or `SiteEntry` form to preview that asset.

For example, a `CSElement` asset named `FiscalNews/Query/ShowHotTopics` expects a value for the `p` variable. If it doesn't receive one, the value of `p` defaults to the object ID of the `Home` page asset. To preview this `CSElement` for a page asset other than the `Home` page, you must pass in the ID of that page asset as the value of the `p` variable with the argument fields in the New or Edit form for that `CSElement` asset.

To specify argument values for previewing a `CSElement` or `SiteEntry` asset:

1. Find the asset and inspect it by clicking the **Inspect** icon (has the letter "i").

2. Scroll to the bottom of the Inspect form. Next to Preview with Arguments, click **Preview**.

3. Enter values for the arguments. You can also select values by double-clicking in the fields and selecting from the drop-down list.

4. Click **Preview**.

5. Click the links that are displayed to preview the pages that are rendered by this `SiteEntry` asset.

18.8 Using Oracle WebCenter Sites Explorer to Create and Edit Element Logic

We don't recommend creating and editing element logic directly from Oracle WebCenter Sites Explorer. However, should you prefer to do so, ensure the validity of your Template and CSElement assets.

Take note of the information in this topic and follow the instructions.

Note:

When a Template (CSElement) asset is created and saved in the WebCenter Sites interface (Template or CSElement form), several important steps are taken that are not taken when you use Oracle WebCenter Sites Explorer:

- The interface seeds your element with stub code that sets compositional dependencies and, if you are using JSP, drops in the appropriate tag library directives for you. Compositional dependencies are described in About Dependencies.

- When you save the Template (CSElement) in the WebCenter Sites interface:

  – The approval system receives information that the asset was changed and can therefore change its approval status.

  – Most importantly, the CacheManager servlet can update the cache (that is, flush pages and pagelets from the WebCenter Sites and Satellite Server caches).

If you choose to work with the CSElement asset in Oracle WebCenter Sites Explorer, be sure that you do not alter the value of the eid variable or accidentally delete it.

A practical reason for using the WebCenter Sites interface is to avoid switching between WebCenter Sites and Oracle WebCenter Sites Explorer, especially if you are mapping asset information (to support template sharing and site replication). Mapping is supported only in the WebCenter Sites interface (in the Map screen of the Template form and in the Map screen of the CSElement form).

This section includes the following topics:

- Creating Templates and CSElements

- Editing Templates and CSElements

18.8.1 Creating Templates and CSElements

If you prefer to use Oracle WebCenter Sites Explorer to code your element logic, follow these steps:

1. Start creating your Template (or CSElement) asset using the Template form (or CSElement form). Start with Open the Template Form (or Open the CSElement Form) and continue sequentially.

2. In Configure the Template's Element (or Configure the Element), select your element type (JSP, XML, or HTML). Do not change the element logic that is auto-generated for you. Also, be sure that you do not alter the value of the tid variable or accidentally delete it.

3. Continue through the form you have chosen until you finish. If you know which keys and asset values you must map, add them to the Map form (in Configure the Map). The same step applies to the CSElement asset.

4. Save the asset.

5. Open Oracle WebCenter Sites Explorer and edit your element. Save your changes.

6. The final step is to re-save your asset in the Template form (or CSElement form). You do not have to change any data in the form, but you must re-save it. This will ensure that no functionality is bypassed.

18.8.2 Editing Templates and CSElements

Any time that you edit an element's logic in the Oracle WebCenter Sites Explorer tool, you need to open and save the template (or CSElement) in the WebCenter Sites interface so that (1) the approval system knows the asset was changed and can change its approval status, and (2) the CacheManager servlet can update the cache.

19

Creating Templates and Wrappers

You need templates to address several different scenarios: to apply to content assets, to render an entire website or a page fragment, to preview content, and so on. WebCenter Sites provides three types of templates: Layout, Pagelet, and Page. Wrappers are SiteEntry assets that contain business logic to be processed before rendering the actual layout template. You use wrappers to access session data for implementing security checks, to determine which locale should be set, to disassemble a friendly URL, and so on.

Topics:

- Working with Templates

- Working with Wrappers

19.1 Working with Templates

You have three types of templates in WebCenter Sites: Layout, Page, and Pagelet. Layout and Page templates can be invoked directly from a browser, but not Pagelet template. Only Layout template can be assigned to an asset. You use Layout template to render an entire website, Page template to preview content, and Pagelet template to render page fragments.

See these topics:

- Layout Templates

- Pagelet Templates

- Page Templates

19.1.1 Layout Templates

A layout template is a template asset where the **Usage** field is set to **Element is used as a layout**. A layout template can be typed or typeless.

A layout template has three main characteristics:

- A layout template can be invoked from a browser

- A layout template can be assigned to an asset

- A layout template typically renders an entire web page

This topic provides details for each of these characteristics and presents use-case scenarios that we will use across multiple sections to demonstrate various concepts. It also explains how you can build a layout template for article assets.

19.1.1.1 A layout template can be invoked from a browser

Given a layout template's pagename, a web page is obtained by calling the Satellite servlet and passing the pagename along with the asset type (c) and asset id (cid) of the content to render:

```
http://localhost:8080/cs/Satellite?pagename=
  <template_pagename>&c=Article&cid=1234567
```

The pagename corresponding to any given template can be found by inspecting a template asset and looking at the **SiteCatalog Pagename** column in the **Site Entries** field.

19.1.1.2 A layout template can be assigned to an asset

Every content asset has template metadata which is viewable by selecting the **Content** tab. This field stores a template name, the possible values include all layout templates applicable to the asset type and subtype.

Figure 19-1   Content Asset with Template Metadata

WebCenter Sites uses the **Template** field when inspecting or editing an asset in Web Mode of the Oracle WebCenter Sites: Contributor interface. WebCenter Sites uses the assigned layout template as the default template to render the asset (this is also the case when simply previewing an asset). In practice, this means that only layout templates can be used to work with assets in Web Mode. Note that previewing does not require a layout template. See Previewing Template, CSElement, and SiteEntry Assets.

In Web Mode, the default layout template can also be assigned by using the Change Layout functionality (either from the toolbar or menu bar), and selecting a layout template visually, using the template picker. In either case, the value of the asset's **Template** field is modified.

Figure 19-2   Change Page Layout Dialog
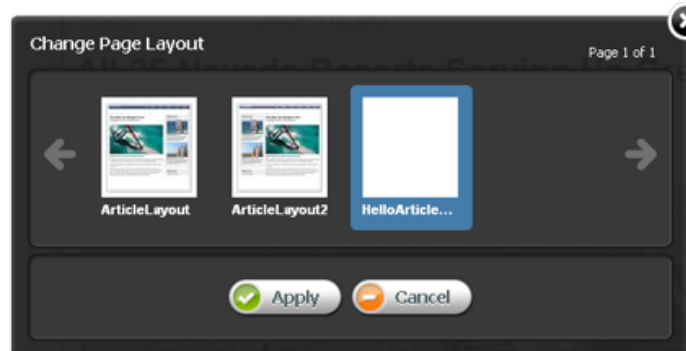
19.1.1.3 A layout template typically renders an entire web page

A standard web page is built with HTML code using <DIV> elements to define divisions within the document, typically a header, footer, side bar, and main area as shown in the diagram below.

Figure 19-3   Sample Layout Template
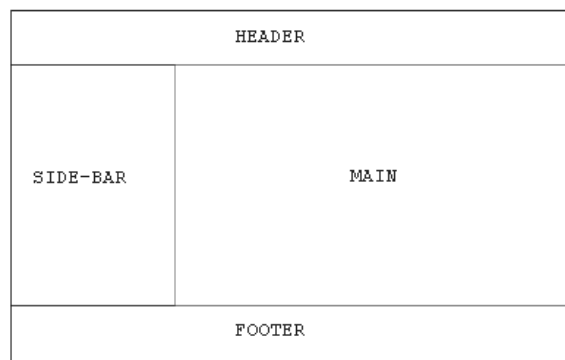
The HTML structure used by the avisports sample site is used in the example below of a standard web page with a header, footer, side bar, and main area.

```
<!DOCTYPE html>
<html>
<head>
</head>
<body class="inner">
  <div id="main">
    <div id="header">
      <!--contains the top menu bar -->
    </div>
    <div id="container">
      <div class="content">
        <!--contains the main area -->
      </div>
      <div class="side-bar">
        <!--contains the side nav bar -->
      </div>
    </div>
    <div id="footer">
      <!--contains the footer -->
    </div>
  </div>
```

```
</body>
</html>
```

An actual JSP page would include tag library directives, an opening and closing `<cs:ftcs>` tag, and the `<render:logdep>` tag, used by the cache manager. See Website Development with Tag Technologies.

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld" %>
<%@ taglib prefix="render" uri="futuretense_cs/render.tld" %>

<cs:ftcs>
<!DOCTYPE html>

<%
// The render:logdep tag is mandatory. It allows the cache manager
// to automatically flush any page/pagelet generated using this
// template ("tid" is automatically populated with the Template
// asset id)
%>

<render:logdep cid='<%=ics.GetVar("tid")%>' c="Template" />

<html>
<head>
</head>
<body class="inner">
  <div id="main">
    <div id="header">
      <!--contains the top menu bar -->
    </div>
    <div id="container">
      <div class="content">
        <!--contains the main area -->
      </div>
      <div class="side-bar">
        <!--contains the side nav bar -->
      </div>
    </div>
    <div id="footer">
      <!--contains the footer -->
    </div>
  </div>
</body>
</html>
</cs:ftcs>
```

The document must be enclosed within <cs:ftcs> tags. This tag creates the WebCenter Sites context, alerting WebCenter Sites that code contained within the opening and closing `<cs:ftcs>` tags will contain WebCenter Sites tags. WebCenter Sites is unaware of any code which falls outside of these tags. For information about `<cs:ftcs>` and `<render:logdep>` tags, see Understanding WebCenter Sites Tags.

19.1.1.4 Use Case 1: Building a Layout Template for Article Assets

Let's define a layout template for article assets.

1. Using Eclipse with the WebCenter Sites Developer Tools plug-in, create a new template in the avisports sample site with these characteristics:

    - **Site**: avisports

- **Name**: HelloArticleLayout

- **Asset Type**: AVIArticle

- **Subtype**: Article

- **Element Usage**: Element is used as a Layout.

- **Element Type**: JSP

- **Root Element**: AVIArticle/HelloArticleLayout

- **Storage Path**: AVIArticle/HelloArticleLayout.jsp

2. Use the following JSP code.

   Note that we are reusing some components written for avisports to render the standard avisports header and footer, and head section, which allows us to import the avisports stylesheets:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld" %>
<%@ taglib prefix="render" uri="futuretense_cs/render.tld" %>

<cs:ftcs>
<!DOCTYPE html>

<render:logdep cid='<%=ics.GetVar("tid")%>' c="Template" />

<html>
<head>
<!--inserts the standard avisports head -->
<render:calltemplate

  tname="/Head"
  args="c,cid"
  style="element" />
</head>
<body class="inner">
  <div id="main">
    <div id="header">
      <!--inserts the avisports navbar -->
      <render:satellitepage
        pagename="avisports/navbar" />
    </div>
    <div id="container" style="height: 350px">
      <div class="content">
        <!--contains the main area -->
      </div>
      <div class="side-bar" style="height: 300px">
        <!--contains the side nav bar -->
      </div>
    </div>
    <div id="footer">
      <!--inserts the avisports footer -->
      <render:callelement

        elementname="avisports/footer" />
    </div>
  </div>
</body>
</html>
</cs:ftcs>
```

3. Open any article asset in a new tab.

4. Assign the HelloArticleLayout template to the asset. You should get a web page like this:

Figure 19-4   Sample Web Page
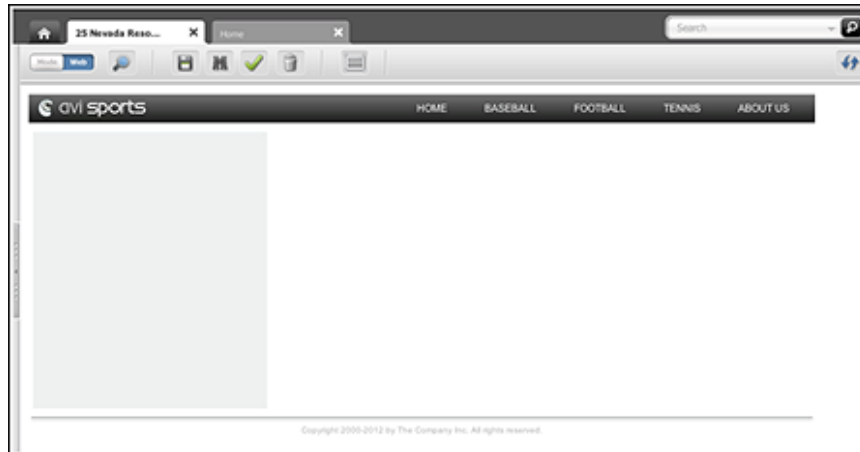
Note that we added some temporary style to the container and side-bar <DIV> elements so they are visible. We will remove them when those contain actual content.

5. Add code to the layout template so it renders actual content. This code renders the headline, post date, related image and body fields.

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"%>
<%@ taglib prefix="ics" uri="futuretense_cs/ics.tld"%>
<%@ taglib prefix="render" uri="futuretense_cs/render.tld"%>
<%@ taglib prefix="assetset" uri="futuretense_cs/assetset.tld"%>

<%@ taglib prefix="dateformat" uri="futuretense_cs/dateformat.tld"%>

<cs:ftcs>
<!DOCTYPE html>

<render:logdep
  cid='<%=ics.GetVar("tid")%>'
  c="Template"/>

<%
// load article content
%>
<assetset:setasset

  name="article"
  type='<%=ics.GetVar("c") %>'
  id='<%=ics.GetVar("cid") %>' />

<%
// fetch the headline, relatedImage, and postDate attributes
// from the database
%>
<assetset:getmultiplevalues
```

```
                   name="article"
                   prefix="article">
                     <assetset:sortlistentry

                       attributename="headline"
                       attributetypename="ContentAttribute" />
                     <assetset:sortlistentry

                       attributename="relatedImage"
                       attributetypename="ContentAttribute" />
                     <assetset:sortlistentry

                       attributename="postDate"
                       attributetypename="ContentAttribute" />
</assetset:getmultiplevalues>


<%
// fetch the body attribute
// body has to be fetched separately, since it is a 'text'
// attribute, and the getmultiplevalues tag does not support
// 'text' attributes
%>
<assetset:getattributevalues

  name="article"
  listvarname="bodyList"
  attribute="body"
  typename="ContentAttribute" />


<%
// read the related AVIImage asset id
%>
<ics:listget
  listname="article:relatedImage"
  fieldname="value"
  output="imageId" />


<%
// read the date value and format it
%>
<ics:listget
  listname="article:postDate"
  fieldname="value"
  output="postDate" />

<dateformat:create
  name="df"
  datestyle="long" />

<dateformat:getdate
  name="df"
  varname="formattedDate"
  valuetype="jdbcdate"
  value='<%=ics.GetVar("postDate") %>' />

<html>
<head>
<!--inserts the standard avisports head -->

<render:calltemplate
```

```
    tname="/Head"
    args="c,cid"
    style="element" />
</head>

<body class="inner">
  <div id="main">
    <div id="header">
      <render:satellitepage
        pagename="avisports/navbar" />
    </div>
    <div id="container">
      <div class="content">
        <div class="top-section section-title">
          <h1>
            <ics:listget
              listname="article:headline"
              fieldname="value" />
          </h1>
          <span class="date">
            <ics:getvar name="formattedDate" />
          </span>
        </div>
        <div class="article post">
          <render:getbloburl
            outstr="imageURL"
            c="AVIImage"
            cid='<%=ics.GetVar("imageId")%>'
            field="largeThumbnail" />
          <img class="photo left"
            src='<%=ics.GetVar("imageURL")%>' />
          <render:stream list="bodyList" column="value" />
        </div>
      </div>
      <div class="side-bar" style="height: 300px">
        <!--contains the side nav bar -->
      </div>
    </div>
    <div id="footer">
      <render:callelement elementname="avisports/footer" />

    </div>
  </div>
</body>
</html>
</cs:ftcs>
```

6. Viewing our asset in Web Mode of the Contributor interface, using HelloArticleLayout, should render a web page with the article detail as shown in the following figure. Note that the side bar is intentionally empty.

Figure 19-5   Sample Web Page Showing Article Detail
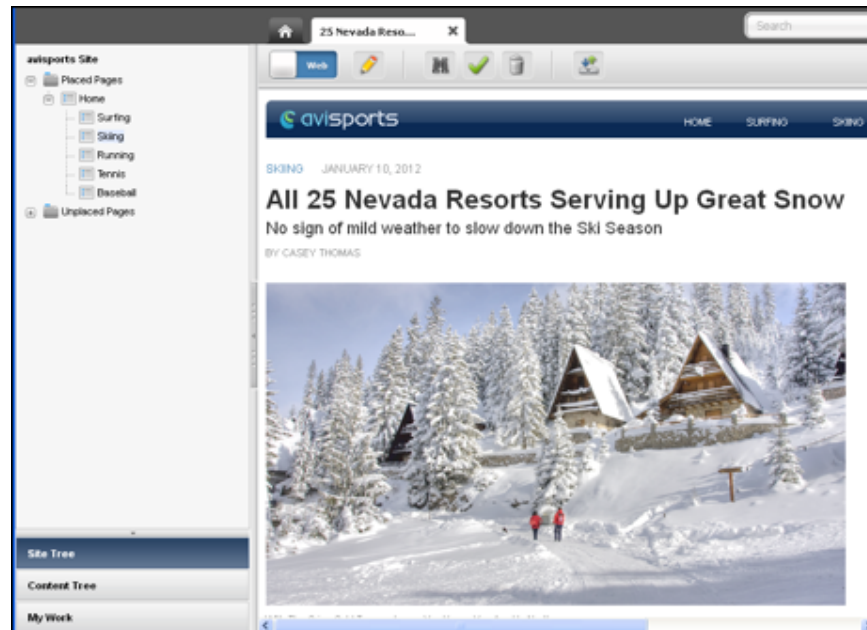
19.1.2 Pagelet Templates

A pagelet template is a template asset for which the Usage field is set to **Element is used within an HTML page**.

A pagelet template has the following characteristics:

- A pagelet template cannot be invoked directly from a browser

- A pagelet template cannot be assigned to an asset

- A pagelet template renders a page fragment

19.1.2.1 A pagelet template cannot be invoked directly from a browser

Although a pagelet template has a pagename, attempting to access a pagelet template directly from a browser using a WCS URL will return a 403 HTTP error code (forbidden).

19.1.2.2 A pagelet template cannot be assigned to an asset

Only layout templates are available in the asset's **Template** field. This means that, these assets cannot be directly previewed using a pagelet template. However, it is possible to set up preview templates, that would give editorial users a simple way to preview pagelet templates.

19.1.2.3 A pagelet template renders a page fragment

A page template renders a web page fragment, not an entire web page. Ideally, a pagelet template represents a reusable page fragment. For instance, a pagelet template could render an article summary block such as shown in this figure:

Figure 19-6   Sample Article Summary Block

Layout templates can then be used to assemble multiple page fragments to produce a complete web page. To maximize reusability, pagelet templates should provide neutral fragments from a look and feel point of view, with CSS stylesheet rules effectively controlling the visual result (based on where a given fragment is used, it would render differently, only by applying a distinct set of stylesheet rules).

19.1.2.4 Use Case 2: Using Pagelet Templates

See Use Case 1: Building a Layout Template for Article Assets for previous steps.

If the code rendering the article detail is meant to be reused in multiple context, it makes sense to extract the code from the layout template, and turn it into a pagelet template.

Let's create the corresponding template asset with the following characteristics:

1. Using Eclipse with the WebCenter Sites Developer Tools plug-in, create the corresponding template in the avisports sample site with these characteristics:

   - **Site**: avisports

   - **Name**: HelloDetail

   - **Asset Type**: AVIArticle

   - **Subtype**: Article

   - **Element Usage**: Element is used within an HTML page.

   - **Element Type**: JSP

   - **Root Element**: AVIArticle/HelloDetail

   - **Storage Path**: AVIArticle/HelloDetail.jsp

2. Let's extract the code responsible for looking up and rendering the article field values inside the `<div class="content">` element and turn it into a separate JSP:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"%>
<%@ taglib prefix="assetset" uri="futuretense_cs/assetset.tld"%>

<%@ taglib prefix="ics" uri="futuretense_cs/ics.tld"%>
<%@ taglib prefix="render" uri="futuretense_cs/render.tld"%>
<%@ taglib prefix="dateformat" uri="futuretense_cs/dateformat.tld"%>
```

```
<cs:ftcs>

<render:logdep
  cid='<%=ics.GetVar("tid")%>'
  c="Template" />

<assetset:setasset

  name="article"
  type='<%=ics.GetVar("c") %>'
  id='<%=ics.GetVar("cid") %>' />

<assetset:getmultiplevalues

  name="article" prefix="article">
    <assetset:sortlistentry

      attributename="headline"
      attributetypename="ContentAttribute" />
    <assetset:sortlistentry

      attributename="relatedImage"
      attributetypename="ContentAttribute" />
    <assetset:sortlistentry

      attributename="postDate"
      attributetypename="ContentAttribute" />
</assetset:getmultiplevalues>


<assetset:getattributevalues

  name="article"
  listvarname="bodyList"
  attribute="body"
  typename="ContentAttribute" />
<ics:listget
  listname="article:relatedImage"
  fieldname="value"
  output="imageId" />
<ics:listget
  listname="article:postDate"
  fieldname="value"
  output="postDate" />
<dateformat:create
  name="df"
  datestyle="long" />
<dateformat:getdate
  name="df"
  varname="formattedDate"
  valuetype="jdbcdate"
  value='<%=ics.GetVar("postDate") %>' />

    <div class="top-section section-title">
      <h1>
        <ics:listget
          listname="article:headline"
          fieldname="value" />
      </h1>
      <span class="date">
```

```
              <ics:getvar name="formattedDate" />
            </span>
        </div>

        <div class="article post">
          <render:getbloburl
            outstr="imageURL"
            c="AVIImage"
            cid='<%=ics.GetVar("imageId") %>'
            field="largeThumbnail" />
          <img class="photo left"
            src='<ics:getvar name="imageURL" />' />
          <render:stream list="bodyList" column="value" />
        </div>
    </cs:ftcs>
```

3. Our layout template can then be modified by simply relying on HelloDetail, invoked using the `<render:calltemplate>` tag:

```
<html>
<head>
  <render:calltemplate

    tname="/Head"
    args="c,cid"
    style="element" />
  </head>
<body class="inner">
  <div id="main">
    <div id="header">
      <render:satellitepage
        pagename="avisports/navbar" />
    </div>
    <div id="container">
      <div class="content">
        <render:calltemplate

          tname="HelloDetail"
          args="c,cid" />
      </div>
      <div class="side-bar" style="height: 300px">
        <!-contains the side nav bar -->
      </div>
    </div>
    <div id="footer">
      <render:callelement

        elementname="avisports/footer" />
    </div>
  </div>
</body>
</html>
```

19.1.3 Page Templates

Page templates are Template assets for which the **Usage** field is set to **Element defines a whole HTML page and can be called externally**.

A page template has the following characteristics:

- A Page Template Can be Invoked from a Browser

- A Page Template Cannot be Assigned to an Asset

- A Page Template Can be Used for Previewing

In this section we discuss each of these characteristics and then demonstrate a practical application of page templates.

19.1.3.1 A Page Template Can be Invoked from a Browser

Like layout templates, a page template can be used to render a web page in a browser by invoking its `pagename` through the Satellite servlet.

19.1.3.2 A Page Template Cannot be Assigned to an Asset

Only layout templates are assignable. Practically speaking, this means that editorial users cannot work in Web Mode of the Contributor interface with a page template, they have to use a layout template.

19.1.3.3 A Page Template Can be Used for Previewing

Previewing an asset is different from using Web Mode in the Contributor interface, in the sense that it only allows editorial users to view an asset, there are no editing capabilities involved. When previewing an asset, WebCenter Sites will use, by default, the layout template set in the asset's **Template** field.

When bringing up the Change Preview Template dialog, the template picker shows the following as valid options:

- All layout templates applicable to the current asset

- All page templates applicable to the current asset

Unlike the Change Layout template picker, selecting a different preview template does not assign this template to the asset's **Template** field. It simply renders the current asset with the selected preview template.

19.2 Working with Wrappers

A wrapper contains business logic to be executed before rendering the actual layout template. It's used when performing actions such as accessing some session data to implement security checks, determining which locale should be set, disassembling a friendly URL, and so on. A wrapper usually does not render any markup, and it's usually uncached.

See these topics on wrappers:

- Creating a Wrapper Page

- Previewing Wrappers

19.2.1 Creating a Wrapper Page

A wrapper is a normal SiteEntry asset, for which the **Wrapper page** flag is set to **Yes**. This specifies that the asset you are creating is a wrapper page. Selecting the **No** flag displays the Pagelet only field. See Creating SiteEntry Assets.

19.2.2 Previewing Wrappers

Depending on the implementation, it might be necessary to execute a wrapper before rendering an asset with a layout template. When you preview an asset, or when you

are working in Web Mode of the Contributor interface, WebCenter Sites will systematically run a wrapper if there is at least one wrapper enabled on the current site.

If a default preview wrapper has been configured for the current editorial site, WebCenter Sites will use this wrapper. Otherwise, it will use the first wrapper available in the list. If several wrappers are available, a different wrapper can be modified by selecting **View** and then **Preview with Wrapper**.

In preview mode, wrappers are especially useful in situations where a layout template requires extra arguments to properly render a web page (for example, a locale argument might be expected) since by default, WebCenter Sites generate a minimal preview URL mainly setting the pagename, with the asset type (c) and asset id (cid) parameters with, respectively, the template pagename, and the type and identifier of the asset to render.

In this case, you can define a preview wrapper, setting extra arguments as required, and then proceed by rendering the layout template. The page name to call is made available in the `childpagename` variable.

```
<cs:ftcs>

<%
// establish appropriate values for required template arguments
%>
<ics:setvar name="foo" value="bar" />

<render:satellitepage
  pagename='<%=ics.GetVar("childpagename")%>'
  args="c,cid,foo" />

</cs:ftcs>
...
```

20

Coding Elements for Templates and CSElements

Elements provide the code that identifies, extracts, and displays your content. Since in a WebCenter Sites system, your content is stored as assets, much of the XML or JSP code in your elements is dedicated to identifying the appropriate asset for the appropriate context, then extracting and displaying that asset's data.

Topics:

- About Dependencies

- About Coding to Log Dependencies

- About Invoking CSElement and SiteEntry Assets

- Coding Elements to Display Basic Assets

- About Coding Elements that Display Flex Assets

- Coding Templates That Display Flex Assets

- Creating URLs for Hyperlinks

- Handling Error Conditions

- Encoding Page Arguments

- What You May Need to Know About Securing Your Site Against XSS Attacks

For information about creating the assets themselves, see Creating Template, CSElement, and SiteEntry Assets.

20.1 About Dependencies

To function properly, your WebCenter Sites system tracks and relies on approval and compositional dependencies. Code your element in a way that the code logs compositional dependencies accurately, and, if you are designing a static site, it sets approval dependencies appropriately, as well.

- **Approval dependencies**are conditions that determine whether an approved asset can be published.

  The approval system calculates the approval dependencies for an asset when it is approved. If there are dependent assets that also have to be approved, the parent asset is not published.

- **Compositional dependencies**, that is, page composition dependencies are dependencies between assets and the pages and pagelets that they are rendered on that determine whether a page needs to be regenerated.

The WebCenter Sites servlet logs compositional dependencies when it renders pages. CacheManager consults the dependency log to determine when to regenerate the cached pages. The Export to Disk publishing method consults the dependency logs to determine when an exported page file must be regenerated.

See these topics:

- The Publishing System and Approval Dependencies

- Page Generation and Compositional Dependencies

20.1.1 The Publishing System and Approval Dependencies

The publishers, editors, and content providers who work on your management system **approve** assets to be published to a target destination. The publishing system then publishes the approved assets automatically, as a background process, according to the schedule that your administration team set up for your WebCenter Sites system.

An asset can be published only if it meets all specified approval dependencies, that is, all associated assets must have been either approved or previously published. If not, the asset is **held** from being published until the dependencies are met: the dependent (related) assets must themselves be approved for publishing to the same destination.

This approval process frees your content and editorial team from the responsibility of manually checking asset dependencies and then publishing a large number of related assets. It also ensures that there can be no broken links on your online site after assets are published.

If an asset is subsequently changed, the asset is no longer considered to be approved, and it must be approved again before it can be re-published.

This section includes the following topics:

- Calculating Approval Dependencies

- Exists vs. Exact vs. None

- Approval Templates for Export to Disk

- Subtypes, Flex Definitions, and Approval Templates

20.1.1.1 Calculating Approval Dependencies

Approval dependencies are recorded at the time the asset is approved. They are written to the ApprovedAssetDeps table in the WebCenter Sites database.

The approval status of an asset is determined by its dependency relationships, which include the approval status of all asset items associated with a particular asset item, and the dependency relationships of those associated items.

The dependency calculation is based on the publishing method:

- For **Export to Disk**, the approval system renders the asset using either the template that is assigned to it or, if there is one specified, the default approval templates for assets of this type. The tags in the template code set approval dependencies that determine the appropriate dependents for the approved asset. The dependent assets must be in an appropriate approval state before the current asset can be published.

- For **Mirror to Server** or **Export Assets to XML**, the approval process examines the data relationships between asset types. Basic assets have associations. Flex assets

have family relationships. Both of these relationships create approval dependencies for these publishing methods. For example, if you approve a flex asset, it will be held from a publishing session unless its parent assets are in an appropriate approval state.

20.1.1.2 Exists vs. Exact vs. None

Approval dependencies can be **exists**, **exact**, and **none**. This section defines each kind of approval type.

You cannot change the approval dependency type for CSElements and SiteEntry assets, embedded links and pagelets, or the Oracle WebCenter Sites: Engage visitor data assets. With the exception of flex attributes, whose dependency type you set when you create the attribute, you also cannot change the approval dependency type for the flex family asset types. For basic asset types, you set the type of approval dependency for their associated assets when you configure the association fields.

When your publishing method is Export to Disk, the tags that set compositional dependencies when pages are rendered also create approval dependencies when the approval system calculates whether an asset can be published. When your code sets approval dependencies on pagelets generated for other assets, you can set the approval type to exists, exact, or none.

> **Note:**
>
> For information about the types of approval dependencies created by the relationships between assets of the various types, see Managing Publishing in *Oracle Fusion Middleware Administering Oracle WebCenter Sites*.

Exists

With an **exists** dependency, the dependent asset must merely exist on the target, the version of the asset does not matter. An **exists** dependency means that an approved parent asset can be published even if a child asset changes (which means that the child asset is no longer approved), if the child asset was previously approved and published to that same destination.

For example, in the following sequence, a collection asset has an **exists** relationship with its ranked children:

- A collection and all of its ranked articles are approved and published to a target.

- One of the ranked articles is edited again, but not approved.

- The collection itself is edited again, approved, and published to the destination.

    The collection is not held back from publishing by the changed but unapproved article, because a prior version of the article exists.

However, in the following example, a collection with an **exists** dependency relationship to its articles cannot be published:

- A collection and all of its ranked articles are approved but not published.

- One of the ranked articles is edited again.

Because the edited article was never published to the destination, it does not yet exist for that destination, which means that the collection cannot be published. The

collection asset is **held** and both the collection and the edited article must be approved before the collection can be published.

The exists approval type is generally useful for links.

Exact

With an **exact** dependency, the dependent asset must be the exact version on the target. No other previously approved version will do. An **exact** dependency means that the parent asset cannot be published if the version of the parent and child assets on the destination do not match.

In the following example, a page asset has an **exact** dependency with its article assets:

1. A page asset and all of its article assets are approved and published to a destination.

2. One of the articles is edited again, but is not re-approved.

3. The page asset is edited and is re-approved.

    The page asset is held, and the resulting form in the WebCenter Sites interface displays a link that points to a list of the assets that must be approved before the page asset can be published. This list shows the article that was edited but not re-approved.

4. The edited article is approved.

    The page asset has been approved and can now be published because the version stamps of the article and the page asset match.

5. Another article asset associated with the page asset is edited.

6. Both the page asset and the edited article asset must be re-approved because the version stamps of the two do not match:

   • The article must be re-approved because it was edited but not yet re-approved.

   • The page asset must be re-approved because it was previously approved with a dependency on a different version of the article.

The exact approval type is generally useful for embedded content.

None

A **none** dependency means that the approved asset can be published no matter what approval state the dependent asset is in. You can set the approval dependency type to `none` by adding the `DEPTYPE` parameter to a tag that sets an approval dependency and setting that parameter to `none`.

Note that setting `DEPTYPE` to `none` affects the approval dependency only. When the Export to Disk process generates the page and invokes the tag, a compositional dependency is logged. But when the approval system invokes the tag during its calculation, no approval dependency is logged.

20.1.1.3 Approval Templates for Export to Disk

When assets are approved for a publishing destination that uses the Export to Disk publishing method, the approval system examines the template assigned to the asset to determine its dependencies.

However, when Export to Disk actually publishes the asset, it does not necessarily use the template that is assigned to the asset. Why? Because the code in another element could determine that a different template is used for that asset in certain cases.

Consider a site that has an asset which can be rendered by several different templates, depending on the context. So when you approve this asset for publishing, which template should the approval process use to determine the dependencies for it? The one that contains the most representative set of dependencies for all of the templates. You may decide to create a special template that contains all the possible dependencies for assets of each type.

What if the template that contains the most representative set of dependencies is not the template that you want to assign to the asset? Set it as the **Default Approval Template** for assets of that type.

You can set Default Approval Templates for each asset type and for each publishing destination. This feature is located in the tree by selecting the **Admin** tab, then selecting **Publishing**, then **Destinations**, then *MyStaticDestinationName*, and then **Set Default Templates**.

> **Note:**
>
> If you specify a default approval template for an asset type on a destination that uses the Mirror to Server publishing method, that template is used when you preview the asset on the Asset Status screen, but not when the asset is approved or published.

20.1.1.4 Subtypes, Flex Definitions, and Approval Templates

If you are using flex assets for a static site, you can assign multiple default approval templates to the flex asset type in the family. You can designate a different default approval template for each flex definition.

For basic assets, the Subtype feature provides a way to further categorize assets of a single asset type. You can use this feature to assign multiple default approval templates for assets of a specific type, based on some other organizing construct.

For example, perhaps the approval template for sports articles should be different than the approval template for world news articles. You can create a sports subtype and a world news subtype for the article asset type and then assign different approval templates for each subtype of the asset type.

You create subtypes for basic assets either in the asset descriptor file when you create the asset type or by using the **Asset Types** option under the **Admin** node in the **General Admin** tree if you decide you need subtypes after the asset types were created. You assign a subtype to an asset by using the New and Edit asset forms. As mentioned, flex assets have subtypes: their flex definitions.

For more information about configuring subtypes for basic assets, and about subtypes in general, see Designing Basic Asset Types.

20.1.2 Page Generation and Compositional Dependencies

Compositional dependencies are recorded in different ways:

- When the Export to Disk publishing method renders a page, it logs compositional dependencies to the appropriate publishing tables. Then, when it's time to publish again, Export to Disk can determine which pages need to be regenerated based on

which assets are being published, it generates all the pages that have logged the assets as compositional dependents.

• When WebCenter Sites renders and caches a page, it logs the dependencies in EhCache at the time a page is rendered and cached. Each row in this table holds the ID of an asset and the cache key or ID of the generated page that the asset was rendered on.

CacheManager and the Page Caches

The CacheManager maintains the WebCenter Sites page caches in EhCache. As assets are changed, it informs EhCache to do flushing and regenerating the appropriate pages basing on the asset dependency. After it makes changes to the WebCenter Sites page cache, the CacheManager communicates that information to all the Satellite Servers participating in your WebCenter Sites system, the co-resident Satellite Server and any remote Satellite Servers that are installed in your system. The Satellite Server applications then update the Satellite page caches.

> **Note:**
>
> If you have the appropriate permissions, you can examine the data using the System Tools for Cache in the Admin tree.

20.1.2.1 CacheManager and Dynamic Publish Sessions

The CacheManager interacts with the publishing system during Mirror to Server publishing session. When a Mirror to Server publishing session ends, the publishing system provides a list of all the IDs of all the assets that were included in the publish operation to the CacheManager servlet on the destination system.

The CacheManager compares that list to the compositional dependencies logged for the pages in the cache to determine which pages and pagelets need to be flushed from the page cache and regenerated. It updates the WebCenter Sites page cache accordingly, and then sends the list of pages to the co-resident and remote Satellite servlets so they can flush those same pages and get new versions from the WebCenter Sites page cache.

20.1.2.2 CacheManager and the Preview Function

When you preview an asset (on the development or management system), the WebCenter Sites interface executes the page name of the template for the asset. ContentServer renders the page, caches the page, and logs the compositional dependencies between the rendered page and the asset.

The CacheManager updates the cached versions of previewed pages when assets are saved. That is, when someone clicks Save, CacheManager compares the object ID of that asset to the compositional dependencies logged for the pages in the cache. It then clears and refreshes the appropriate pages in the page cache and communicates the information about the changed pages to the Satellite servlets.

20.2 About Coding to Log Dependencies

In your element, remember to include code that logs dependencies accurately. Several tags can log compositional dependencies. On processing a tag, WebCenter Sites logs a dependency between the rendered page and the asset.

For a static site using the Export to Disk publishing method, the tags that log compositional dependencies can also log approval dependencies. When an asset is

approved, the approval system renders that asset to determine whether it can be published. It logs the results of these tags to the `ApprovedAssetDep` table unless the tag sets the approval dependency type to none. See Exists vs. Exact vs. None.

The topics that follow present the tags that log dependencies in alphabetic order. For more information about these and any other tag, see the *Oracle Fusion Middleware Tag Reference for Oracle WebCenter Sites Reference*.

Topics:

- ASSET.LOAD and asset:load

- The ASSETSET (assetset) Tag Family

- RENDER.GETPAGEURL and render:getpageurl

- RENDER.LOGDEP (render:logdep)

- RENDER.FILTER and render:filter

- RENDER.UNKNOWNDEPS and render:unknowndeps

20.2.1 ASSET.LOAD and asset:load

When WebCenter Sites executes an `ASSET.LOAD` tag (or `asset:load`), it automatically logs a compositional dependency for the asset that is loaded. For example:

```
<ASSET.LOAD TYPE="Page" NAME="target" FIELD="name" VALUE="Home"/>
```

This line of code marks a compositional dependency between the page asset named Home and the rendered page that is displaying this asset.

Setting the Approval Dependency Type

When an asset is approved for an Export to Disk destination and the approval system renders this tag, the tag also logs an approval dependency between the assets that are in play.

By default, the approval dependency for `ASSET.LOAD` is set to `exact`. You can set the dependency to `exists` or to `none` by using the `DEPTYPE` parameter. For example:

```
<ASSET.LOAD TYPE="Page" NAME="target" FIELD="name" VALUE="Home" DEPTYPE="exists"/>
```

20.2.2 The ASSETSET (assetset) Tag Family

You use the `ASSETSET` tag family to create a set of one or more flex assets. The following tags create assetsets and define compositional dependencies for the assets in the set:

```
ASSETSET.SETASSET and assetset:setasset
ASSETSET.SETEMPTY and assetset:setempty
ASSETSET.SETLISTEDASSETS and assetset:setlistedassets
ASSETSET.SETSEARCHEDASSETS and assetset:setsearchedassets
```

When an asset from the assetset is rendered, the compositional dependency is logged.

The first three tags define the following compositional dependencies:

- A dependency between each flex asset in the assetset and the rendered page.

- A dependency between the flex asset's parents and the rendered page. Because flex assets inherit values from their flex parent assets, a change to a parent can mean a change to the flex asset and that means the pages that hold the asset may no longer be accurate.

The fourth tag, `assetset:setsearchedassets`, creates an assetset from the results of a search state. Search states are queries, which means there is no way to predict the identities of the assets in the set. Therefore, the `ASSETSET.SETSEARCHEDASSETS` tag defines the compositional dependency as unknown. When a compositional dependency is unknown, it means the page must be regenerated during each Export to Disk publishing session and updated in the page caches after each Mirror to Server publishing session, whether it needs it or not.

If you have a search state that describes a fixed set of assets whose identities will not change, you instruct WebCenter Sites to set compositional dependencies for the assets in the assetset by setting the optional `fixedlist` property to true.

For example:

```
<assetset:setsearchedassets name=as assettypes=Products constrain=ss
fixedlist=true />
```

This example defines that there is a compositional dependency between each product asset in the assetset named as and the rendered page.

See Assetsets and Searchstate Objects.

20.2.2.1 Setting the Approval Dependency Type

If you are using flex assets for a static site, be aware that when the approval system invokes an `assetset` tag, the approval dependency type is set to `none` by default.

To change this value to `exists` or `exact`:

- Use the `deptype` parameter. For example:

  ```
  <assetset:setsearchedassets name=as assettypes=Products constrain=ss
  fixedlist=true deptype=exists />
  ```

  Setting an approval type for the `assetset:setsearchedassets` tag is meaningful only if the `fixedlist` parameter is set to `true`.

20.2.3 RENDER.GETPAGEURL and render:getpageurl

The `RENDER.GETPAGEURL` tag creates a URL for assets that are not blobs. This tag logs an exists *approval* dependency, but not a compositional dependency, between the asset being approved (rendered) and the asset referred to by the tag. This means that it creates a dependency only when your publishing method is Export to Disk.

In this example, the template assigned to article ABC has the following code in it:

```
<RENDER.GETPAGEURL PAGENAME="<site_name>/Page/Home"
cid="Variables.pageid"
c="Page"
OUTSTR="referURL"/>
```

That code fragment both creates a URL (that is returned in the variable created by the `OUTSTR` parameter) and logs an **exists** approval dependency between the asset identified in the `cid` variable and article ABC.

Then, when article ABC is approved, the page identified by the `cid` variable must either be approved or must have been published or article ABC is held from being published.

20.2.4 RENDER.LOGDEP (render:logdep)

There are several situations in which your code can obtain an asset's data without actually loading the asset. When this is the case, be sure to log the compositional dependency yourself with the `render:logdep` tag.

Example 1

When you call a CSElement from a Template asset or other CSElement asset, you do not load the asset to determine the identity of the element file to execute. Instead, you use the `RENDER.CALLELEMENT` or `render:callelement` tag and invoke the element directly by name. For example:

```
<render:callelement name=<site_name>/Common/HeaderText/>
```

Because you didn't use the `asset:load` tag to access the CSElement, the compositional dependency between the CSElement asset and the page it is being rendered on is not automatically logged for you. Instead, you must set it yourself.

At the beginning of the element for each CSElement asset, you include the following line of code:

```
<render:logdep cid="Variables.eid" c="CSElement"/>
```

At the beginning of the element for a Template asset, the `render.logdep` statement would be as follows:

```
<render:logdep cid="Variables.tid" c="template"/>
```

Note that if you use the CSElement form or the template form in the WebCenter Sites interface to start coding the element, WebCenter Sites automatically includes an appropriate `render:logdep` statement in the stub code that it seeds into the element for you.

Example 2

For basic assets, when you use an `ASSET.LOAD` tag on a parent asset (basic asset) and then use an `ASSET.CHILDREN` tag, you have access to the children assets' data without having to load it. In this case, you should include a `RENDER.LOGDEP` statement to log the compositional dependency.

For example:

```
<ASSET.CHILDREN NAME="PlainListCollection" LIST="theArticles"
OBJECTTYPE="Article" ORDER="nrank" CODE=-/>
<LOOP LIST="theArticles">
<RENDER.LOGDEP cid="theArticles.id" c="Article"/>
...
```

20.2.4.1 Setting the Approval Dependency Type

When an asset is approved for an Export to Disk destination and the approval system invokes this tag, the tag also creates an exact approval dependency between the asset and the rendered page.

- You can change the approval dependency type to exists or none by setting the `DEPTYPE` argument, as in the following example:

```
<RENDER.LOGDEP cid="theArticles.id" c="Article" DEPTYPE="exists"/>
```

20.2.5 RENDER.FILTER and render:filter

You use the `RENDER.FILTER` tag for lists of assets created by queries. This tag filters out any unapproved assets from a list or a query. It also sets a compositional dependency of unknown.

You use this tag when you do not want an approved asset that has an approval dependency on the results of a query (a collection or query asset, for example) to be held from being published when there are unapproved assets in the list that is returned by the query. For example, say that the element is coded to provide appropriate formatting for any number of article assets that are passed to it so it doesn't matter if only two of the five articles included in a collection cannot be published. Because this tag tells Export to Disk to filter out the unapproved assets, a page using the query can be published while the unapproved assets remain unpublished.

You might use this tag in the following places:

- Templates for query assets

- Templates for collection assets

- `SELECTTO` statements and `EXECSQL` queries

For example:

```
<RENDER.FILTER LIST="ArticlesFromWireQuery" LISTVARNAME="ArticlesFromWireQuery"
LISTIDCOL="id"/>
```

20.2.6 RENDER.UNKNOWNDEPS and render:unknowndeps

The `RENDER.UNKNOWNDEPS` tag signals that there are dependent assets but that there is no way to predict the identities of those assets because they came from a query or change frequently. This tag logs a compositional dependency of unknown for the rendered page. This tag does not set an approval dependency for the Export to Disk publishing method.

When a compositional dependency is set to unknown, it means the page must be regenerated during each Export to Disk publishing session and updated in the page caches after each Mirror to Server publishing session, whether it needs it or not.

> **Note:**
>
> You must use this tag carefully because the more pages that must be regenerated, the longer it takes to publish your site.

You use this tag to cover those coding situations in which you truly cannot determine what the dependent assets might be. For example, queries are dynamic and can retrieve a different resultset every time they are run. When you use queries of any kind, query assets, `SELECTTO` statements, `EXECSQL`, and so on, you should use the `RENDER.UNKNOWNDEPS` tag.

20.3 About Invoking CSElement and SiteEntry Assets

From a coding point of view, you are not interested in the CSElement or SiteEntry as an asset, but in the element or page entry that the asset represents. So, write a code that directly invokes the element or page entry with the appropriate tag.

If a CSElement does not have a corresponding SiteEntry asset (which means its output is cached according to the cache criteria set for the calling page), or, if you don't need a separate pagelet at this invocation, you invoke it by name with the `RENDER.CALLELEMENT` (`render:callelement`) tag. For example:

```
<render:callelement name="FiscalNews/Common/SetHTMLHeader"/>
```

When CSElement does have a corresponding SiteEntry asset, you invoke the element by calling the page name of its SiteEntry asset with the `RENDER.SATELLITEPAGE` (`render:satellitepage`) tag. For example:

```
<render:satellitepage pagename="FiscalNews/Pagelet/Common/SiteBanner"/>
```

> **Note:**
>
> When you use Oracle WebCenter Sites Explorer to examine `SiteCatalog` and `ElementCatalog` entries, they are presented as folders and subfolders that visually organize the pages and pagelets. However, these entries are simply rows in a database table, there is no actual hierarchy. Therefore your code must always call a page entry or an element entry by its entire name. You cannot use a relative path.
>
> Additionally, the chain of called elements should not be more than 20 levels deep. Otherwise, the system will perform poorly when displaying the assets.
>
> Also, if you edit using Oracle WebCenter Sites Explorer, save the asset in the asset's editorial form (in the WebCenter Sites interface) to ensure that the cache is updated to reflect your edits. (Oracle WebCenter Sites Explorer does not automatically update the cache.)

20.4 Coding Elements to Display Basic Assets

Get a deeper understanding of how the asset type—for whose template you're writing an element code—is designed. With just a preliminary understanding of data and site design, you may find it hard to re-code templates that display updated assets.

To help you develop a thorough understanding, this topic discusses:

- What you should keep in mind when you code templates for basic asset types.

- Code fragments and examples for various situations. One situation is managing dependencies between assets for correctly calculating approval for static sites, and clearing the page cache for dynamic sites, when it's appropriate.

Before you begin, be sure to read the chapters in the Programming Basics section of this book, especially Website Development with Tag Technologies.

For information about the tags used in the code examples, see the *Oracle Fusion Middleware Tag Reference for Oracle WebCenter Sites Reference*.

See Template Element Examples for Basic Assets.

Topics:

20.4.1 Assets That Represent Simple Content

Template elements for content assets generally extract one specific article, advertising copy, special offer, image, and so on from the database, then obtain information from the relevant fields such as headline, body, and byline (for example), and then display that information online.

Consider the following simple template element designed for an article asset:

```
<?xml version="1.0" ?>
<!DOCTYPE FTCS SYSTEM "futuretense_cs.dtd">
<FTCS Version="1.1">
<!-- Article/VeryBasic
-
- INPUT
- Variables.c - asset type (Article)
- Variables.cid - id of the asset to display
- Variables.tid - template used to display the page(let)
- OUTPUT
-
-->
<!-- log the template as a dependent of the pagelet being rendered, so changes to
the template will force regeneration of the page(let) -->

<IF COND="IsVariable.tid=true">
<THEN>
<RENDER.LOGDEP cid="Variables.tid" c="Template"/>
</THEN>
</IF>
<!-- asset load will mark the asset as an 'exact' dependent of the pagelet being
rendered -->

<ASSET.LOAD NAME="anAsset" TYPE="Variables.c" OBJECTID="Variables.cid"/>

<!-- get all the primary table fields of the asset -->

<ASSET.SCATTER NAME="anAsset" PREFIX="asset"/>

<!-- display the description -->
<ics.getvar name=asset:description/>

<!-- display the contents of the urlbody file -->

<ics.getvar name="asset:urlbody" encoding="default"
output="bodyvar"/>
<RENDER.STREAM VARIABLE="bodyvar" /><br/>
```

```
</FTCS>
```

The code in this template does the following things:

- Logs a compositional dependency between the Template asset and the page being rendered with the element with the `RENDER.LOGDEP` tag.

- If the approval system is evaluating this code for an Export to Disk target, logs an approval dependency.

- Loads the article asset with an `ASSET.LOAD` tag, which logs a compositional dependency between the article asset and the page being rendered.

- Extracts all the values from all the fields of the article with an `ASSET.SCATTER` tag.

- Displays the contents of the description column with a `CSVAR` tag. The description column corresponds to the **Headline** field in the **New** or **Edit** article forms in the WebCenter Sites interface.

- Displays the contents of the `urlbody` column with the `ics.getvar` and `RENDER.STREAM` tags. The `urlbody` column corresponds to the **Headline** field in the **New** or **Edit** article forms in the WebCenter Sites interface.

Notice the difference in the code that displays the value from the `description` column and the code that displays the value from the `urlbody` column. The `urlbody` column can contain embedded links and whenever a field can contain embedded links, you ensure that the links are rendered correctly by using the `RENDER.STREAM` tag rather than the `CSVAR` tag.

20.4.2 Associations

You identify the assets that are associated with other assets through association fields with the `ASSET.CHILDREN` tag. To specify which associated asset, you use the `CODE` parameter to specify the association field.

For example, say that the following code fragment is inserted right before the `</FTCS>` tag in the preceding example:

```
<!-- display the Main Image -->
<ASSET.CHILDREN NAME="anAsset" LIST="associatedImage"
CODE="MainImage"/>
<IF COND="IsList.associatedImage=true">
<THEN>
<RENDER.SATELLITEPAGE PAGENAME="FiscalNews/ImageFile/TeaserSummary"
ARGS_cid="associatedImage.oid"/>
</THEN>
</IF>
```

The code in this fragment does the following things:

- Extracts the `imagefile` asset that is specified in the **Main Image** field for this article asset (named `anAsset`) with the `ASSET.CHILDREN` tag and the `CODE` parameter set to MainImage.

- Passes the identity of that `imagefile` to the page entry for the `TeaserSummary` template with the `RENDER.SATELLITEPAGE` tag. The page entry is identified with the `PAGENAME` parameter and the `imagefile` is identified with the `ARGS_cid` parameter. The `TeaserSummary` template than renders the

imagefile into a pagelet and passes the pagelet back to this page, where it is displayed with the article.

20.4.3 ImageFile Assets or Other Blob Assets

The imagefile asset type stores uploaded image files. In other words, the imagefile asset type is a **b**inary **l**arge **ob**ject (blob), served from the WebCenter Sites database. You use the BlobServer servlet to serve and display imagefiles and other blobs.

A template element for an imagefile or other blob can use the RENDER.SATELLITEBLOB tag to create and return an HTML tag that tells the browser how to access the blob and how to format and display it. If you need a BlobServer URL only, without it being embedded in an HTML tag, you can use the RENDER.GETBLOBURL tag.

For more information about coding links to blobs, see Creating URLs for Hyperlinks.

20.4.4 Basic Assets That Can Have Embedded Links

The **Body** field of the Article asset and other assets that have fields with a data type of TEXTAREA allow editors to create embedded hyperlinks within the text field. To ensure that these links are rendered properly, you can use the RENDER.STREAM tag to retrieve the contents of the field, as shown in the following example:

```
<asset:load name="TestArticle" type="<%=ics.GetVar("c")%>"
 objectid='<%=ics.GetVar("cid")%>' />
<asset:scatter name="MainArticle" prefix="articleAsset" />
<!-- display the contents of the urlbody file -->

<ics:getvar name="articleAsset:urlbody" encoding="default"
 output="bodyvar"/>
<render:stream variable="bodyvar"/><br/>
```

If Web Mode is enabled on your management system, note that the insite:edit tag also manages embedded links appropriately when it retrieves the contents of a field that has embedded links in it.

20.4.5 Collections

Templates for collection assets typically extract the assets in the collection from the database with an ASSET.CHILDREN tag. For example:

```
<ASSET.LOAD TYPE="Variables.c" OBJECTID="Variables.cid"
NAME="PlainListCollection"/>
<ASSET.SCATTER NAME="PlainListCollection" PREFIX="asset"/>
<ASSET.CHILDREN NAME="PlainListCollection" LIST="theArticles"
OBJECTTYPE="Article"/>
```

After the children are identified, the template code can then display parts of these assets in a list on a rendered page.

Sometimes the template for a collection is coded to handle the first item in the collection differently than the rest. You can single out the highest ranking asset in a collection by coding the element to order the items in the list according to their rank, as shown here:

```
<ASSET.CHILDREN NAME="HomePageStories" LIST="theArticles"
OBJECTTYPE="Article" ORDER="nrank"/>
```

This section includes the following topics:

- Collection Templates and Approval Dependencies

- Collection Templates and Compositional Dependencies

20.4.5.1 Collection Templates and Approval Dependencies

When your publishing method is Export to Disk, you can use the `RENDER.FILTER` tag in your collection templates. This tag filters out any unapproved assets from the collection both when the approval dependencies are calculated and when the publish process renders the site.

The following code fragment illustrates this tag:

```
<ASSET.LOAD TYPE="Variables.c" OBJECTID="Variables.cid"
NAME="StoryListCollection"/>
<ASSET.SCATTER NAME="StoryListCollection" PREFIX="asset"/>
<ASSET.CHILDREN NAME="StoryListCollection" LIST="theArticles"
ORDER="nrank" CODE="-"/>

<!-- Get only the articles that are approved for export -->

<RENDER.FILTER LIST="theArticles"
LISTVARNAME="ApprovedArticles"
LISTIDCOL="oid"/>

<!-- Display only the articles that are approved-->

<IF COND="IsList.ApprovedArticles=true">
<THEN>
<LOOP LIST="ApprovedArticles">
<RENDER.SATELLITEPAGE
PAGENAME="<site_name>/Article/Summary"
ARGS_cid="ApprovedArticles.oid"
ARGS_p="Variables.p"/>
</LOOP>
</THEN>
</IF>
```

20.4.5.2 Collection Templates and Compositional Dependencies

In the preceding code example that illustrates the `RENDER.FILTER tag`, the ID of each of the child assets in the collection is passed to the Summary template.

The first line of code in the Summary template is an `ASSET.LOAD` statement, which means that the dependency between article asset that it loads and the page that is rendered with the Summary template is logged.

If the code in the template for the collection also formats the child articles, you must carefully consider the code and determine whether you have to log the dependency with the `RENDER.LOGDEP` tag.

For example, when you use the `OBJECTYPE` parameter in an `ASSET.CHILDREN` tag, the resulting list is a join of the `AssetRelationTree` table and the asset table for the type specified and includes information from both tables, as in the following example:

```
<ASSET.CHILDREN NAME="StoryListCollection" LIST="theArticles"
OBJECTTYPE=Article ORDER="nrank" CODE="-"/>
```

You can then access the children asset's information without using subsequent `ASSET.LOAD` tags. If you do, be sure to include the `RENDER.LOGDEP` tag for each

child so that the compositional dependencies between those assets and the rendered page can be tracked correctly.

For another example, see Coding Links to the Article Assets in a Collection Asset.

20.4.6 Query Assets

Query assets can execute SQL code or they can run an element that contains query code. You use them in such applications as collections and page assets.

- You build a collection by running a query in the Build Collection form and then selecting and ordering the assets you want from the resulting list. The collection is a static list of assets selected from the query's resultset.

- You select queries for a page asset either through unnamed relationships or through associations. You select queries for assets like articles through associations.

  In these cases, the page or article assets do not themselves invoke the query; you code the query template element to invoke a standard WebCenter Sites element called `OpenMarket/Xcelerate/AssetType/Query/ExecuteQuery`. This element runs the query asset when the page asset or article asset is rendered.

Elements for query templates invoke the `ExecuteQuery` element and typically include code that loops through the items returned in the list object that the query created, extracts bits of information from those items, and then displays it.

The following example loads a query asset and passes it to the `ExecuteQuery` element:

```
<ASSET.LOAD TYPE="Query" NAME="Wirefeed" OBJECTID="Variables.id"/>
<CALLELEMENT NAME="OpenMarket/Xcelerate/AssetType/Query/
ExecuteQuery">
<ARGUMENT NAME="list" VALUE="ArticlesFromWireFeed"/>
<ARGUMENT NAME="assetname" VALUE="WireFeed"/>
<ARGUMENT NAME="ResultLimit" VALUE="-1"/>
</CALLELEMENT>
```

Queries and Compositional Dependencies

The first line of code in the `ExecuteQuery` element is a `RENDER.UNKNOWNDEPS` tag, which alerts the Export to Disk publishing method and the CacheManager on a dynamic delivery system that the assets that will be retrieved by the query cannot be predicted and, therefore, no dependencies can be calculated and logged.

If you are using any other kind of query, for example, a `SELECTTO` statement, `CALLSQL`, or `EXECSQL`, you should include the `RENDER.UNKNOWNDEPS` tag.

Additionally, in the element that a query-generated list of assets is returned to, you must use the `RENDER.FILTER` tag if you are using the Export to Disk publishing method, as in the following example:

```
<CALLELEMENT NAME="OpenMarket/Xcelerate/AssetType/Query/ExecuteQuery">
<ARGUMENT NAME="list" VALUE="ArticlesFromTheQuery"/>
<ARGUMENT NAME="assetname" VALUE="PlainListQuery"/>
<ARGUMENT NAME="ResultLimit" VALUE="5"/>
</CALLELEMENT>

<!-- On export - filter out un-approved assets -->
<RENDER.FILTER LIST="ArticlesFromTheQuery" LISTVARNAME="ArticlesFromTheQuery"
LISTIDCOL="id"/>
```

```
<if COND="ArticlesFromTheQuery.#numRows!=0">
<then>
<LOOP LIST="ArticlesFromTheQuery">
<RENDER.GETPAGEURL PAGENAME="FiscalNews/Article/
Variables.ct"
cid="ArticlesFromTheQuery.id"
c="Article"
p="Variables.p"
OUTSTR="referURL"/>
<A class="wirelink" HREF="Variables.referURL"
REPLACEALL="Variables.referURL"><ics.listget listname=ArticlesFromTheQuery
fieldname=subheadline/>
</A><P/>
```

For another example, see Coding Templates for Query Assets.

20.4.7 Page Assets

Templates for page assets generally contain the following kinds of code:

- The framework for the page asset when it is a rendered page

- The logic for obtaining the content for the rendered page

- The logic for links to other rendered pages

The templates for content assets contain the formatting code for individual pieces of content. The page templates invoke the templates for the other assets, receive formatted assets from those template elements, and then place the formatted assets into the context of the page framework.

The following is the code for a simple template that formats a page asset:

```
<?xml version="1.0" ?>
<!DOCTYPE FTCS SYSTEM "futuretense_cs.dtd">
<FTCS Version="1.1">
<!-- Page/CollectionsAndQuery
-
- INPUT
- Variables.c - asset type (Page)
- Variables.cid - id of the asset to display
- Variables.tid - template used to display the page(let)
- OUTPUT
-
-->

<!-- log the template as a dependent of the pagelet being rendered, so changes to
the template will force regeneration of the page(let) -->

<IF COND="IsVariable.tid=true">
<THEN>
<RENDER.LOGDEP cid="Variables.tid" c="Template"/>
</THEN>
</IF>

<!-- asset load will mark the asset as an 'exact' dependent of the pagelet being
rendered -->

<ASSET.LOAD NAME="anAsset" TYPE="Variables.c"
OBJECTID="Variables.cid"/>

<!-- get all the primary table fields of the asset -->
```

```
<ASSET.SCATTER NAME="anAsset" PREFIX="asset"/>

<!-- get a list of id's of the child assets in the collection in order of their rank
-->

<!-- get the WireFeed query -->

<ASSET.CHILDREN NAME="HomeTextPage" LIST="WireFeedStories"
CODE="WireFeed"/>
<IF COND="IsList.WireFeedStories=true">
<THEN>
<RENDER.GETPAGEURL PAGENAME="<site_name>/Query/WireFeedFrontText"
cid="WireFeedStories.oid"
c="Query"
p="Variables.asset:id"
OUTSTR="referURL"/>
<P>
<A HREF="Variables.referURL"
REPLACEALL="Variables.referURL">From the Wires...</A>

</P>
<RENDER.SATELLITEPAGE PAGENAME="<site_name>/Query/WireSummaryText"
ARGS_cid="WireFeedStories.oid"
ARGS_ct="WireStoryText"
ARGS_p="Variables.asset:id"/>
</THEN>
</IF>
</FTCS>
```

The code in this example does the following:

- Logs a compositional dependency between the Template asset and the page being rendered with a RENDER.LOGDEP tag.

- Loads the page asset with an ASSET.LOAD tag, which logs a compositional dependency between the article asset and the page being rendered.

- Extracts the WireFeed query with an ASSET.CHILDREN tag and the CODE parameter set to WireFeed.

- Obtains a URL for a page that will display the stories from the WireFeed query with the RENDER.GETPAGEURL tag. The PAGENAME parameter specifies the page entry of the template to use to create that page and also determines part of the URL. The OUTSTR parameter creates a variable named referURL to hold the URL that RENDER.GETPAGEURL creates.

- Uses the URL from the referURL variable to build an <A HREF> link to the page.

- Passes the identity of the query asset to the page entry for the WireSummaryText template. The WireSummaryText template then creates a pagelet that displays the summary text from each article returned by the Wire Feed query and passes the pagelet back to this page, where it is displayed.

20.5 About Coding Elements that Display Flex Assets

WebCenter Sites provides ASSETSET and SEARCHSTATE tag families for coding elements that display flex assets.

When you code templates for basic assets, you use the WebCenter Sites ASSET tag family. For example, when you want to extract and display a basic asset, you use the ASSET.LOAD tag, a tag that extracts data from the primary storage table for that asset type. But, the database schema for flex assets is different than that for basic assets, so WebCenter Sites provides these tag families for flex assets that you use in place of the ASSET tags:

- ASSETSET. You use this tag family to specify a set of one or more flex assets.

- SEARCHSTATE. You use this tag family to create search constraints that filter the assets in an assetset.

> **Note:**
>
> The ASSET.LOAD tag will load a flex asset for you. However, using the ASSET.LOAD tag with flex assets is not supported: the code cannot be upgraded, and extracting the asset in this way is slower by orders of magnitude than using the ASSETSET tag family.

When you use the flex asset model to represent your content, your online site will use a mixture of flex and basic assets because the page asset type (which you are likely to use) is a basic asset type.

Topics:

- Assetsets

- Searchstate Objects

- Assetsets, Searchstates, and Flex Attribute Asset Types

- Scope

20.5.1 Assetsets

An **assetset** is a group of one or more flex assets or flex parent assets. You use the ASSETSET tags to create the set of assets and to extract the attribute values that you want to display.

You can retrieve the following information from an assetset:

- The values for one attribute for each of the flex assets in the assetset.

- The values for multiple attributes for each of the flex assets in the assetset.

- A list of the flex assets in the assetset.

- A count of the flex assets in the assetset.

- A list of unique attribute values for an attribute for all flex assets in the assetset.

- A count of unique attribute values for an attribute for all flex assets in the assetset.

You can create assetsets that include flex assets of multiple types, but only if those flex assets use the same flex attribute asset type.

The most commonly used ASSETSET tags are:

```
ASSETSET.SETASSET
ASSETSET.SETSEARCHEDASSETS
```

```
ASSETSET.GETMULTIPLEVALUES
ASSETSET.GETATTRIBUTEVALUES
ASSETSET.GETASSETLIST
ASSETSET.SORTLISTENTRY ...
```

All of the `ASSETSET` tags are described in the *Oracle Fusion Middleware Tag Reference for Oracle WebCenter Sites Reference* and several of them are used in the code samples in this chapter. For information about compositional dependencies and the assetset tags, see The ASSETSET (assetset) Tag Family.

20.5.2 Searchstate Objects

With searchstate objects, you can obtain the IDs of the flex assets that you want to display.

A **searchstate** is a set of search constraints based on the attribute values held in the `_Mungo` table for the flex asset type. You **apply** searchstates to **assetsets**.

You build a searchstate by adding or removing constraints to narrow or broaden the list of flex assets that are described by the searchstate. For example, if you have a Lighting site whose purpose is to sell lighting supplies, you can use searchstates to create drill-down searching features that visitors use to browse through the site's product catalog.

An unconstrained searchstate applied to an assetset creates an unfiltered list of all the assets of that type. For example, the following code sample would create an assetset that contains all the products in the Lighting site's product catalog:

```
<SEARCHSTATE.CREATE NAME=nolimits/>
<ASSETSET.SETSEARCHEDASSETS NAME=unconstrainedAssetSet
CONSTRAINT=nolimits ASSETTYPES=Products/>
```

To narrow the number of products in the assetset, you add constraints. For example, the following code sample would create an assetset that contains only the 40-watt light bulbs from the catalog:

```
<SEARCHSTATE.CREATE NAME=lightbulbs/>
<SEARCHSTATE.ADDSIMPLESTANDARDCONSTRAINT NAME=lightbulbs
 ATTRIBUTE=wattage VALUE=40/>
<ASSETSET.SETSEARCHEDASSETS NAME=40WattLightbulbs
CONSTRAINT=lightbulbs ASSETTYPES=Products/>
```

A constraint is a filter (restriction) that can be based on the value of an attribute or it can be based on another searchstate, which is called a nested searchstate.

A searchstate can search either the `_Mungo` table for the asset type database or the attribute indexes created by a search engine for that asset type. This means that you can mix database and rich-text (full-text through an index) searches in the same query. To apply a constraint against a search engine index, use the `SEARCHSTATE.ADDRICHTEXTCONSTRAINT` tag.

Note:

Using SQL to query the flex asset database tables instead of using the `SEARCHSTATE` tag family is not supported.

The most commonly used `SEARCHSTATE` tags are as follows:

```
SEARCHSTATE.CREATE
SEARCHSTATE.ADDSTANDARDCONSTRAINT
SEARCHSTATE.ADDSIMPLESTANDARDCONSTRAINT
SEARCHSTATE.ADDRANGECONSTRAINT
SEARCHSTATE.ADDRICHTEXTCONSTRAINT
SEARCHSTATE.TOSTRING
SEARCHSTATE.FROMSTRING
```

All of the SEARCHSTATE tags are described in the *Oracle Fusion Middleware Tag Reference for Oracle WebCenter Sites Reference* and several of them are used in the code samples in this chapter.

20.5.3 Assetsets, Searchstates, and Flex Attribute Asset Types

Because searchstates filter select assets based on attribute values, and assetsets are created by applying searchstates to the assets in the database, only those flex asset types that share the same attribute asset type can be included in the same assetset.

For example, if a site has a content attribute that is shared by two flex asset types such as a flex article asset type and a flex image asset type, then you can create an assetset with both flex articles and flex images in it. However, if the site also has a product asset type that uses a product attribute instead of the content attribute, you could not create an assetset that contains both flex articles and product assets or both flex images and product assets.

20.5.4 Scope

The scope of assetsets and searchstates is local; that is, they exist only for the current element (rendered page).

When you want to maintain the existing searchstate, you can use the SEARCHSTATE.TOSTRING tag to convert it to a string and then include that string as an argument in the URL for the next page.

For example:

```
<SEARCHSTATE.TOSTRING NAME=ss VARNAME=stringss/>
<RENDER.SATELLITEPAGE
pagename= SiteName/Products/Example
ARGS_search=Variables.stringss/>
```

And then, in the root element of this example page that receives the string, you code another searchstate:

```
<SEARCHSTATE CREATE NAME=ss/>
```

And unpack the string that was passed to the example element with a SEARCHSTATE.FROMSTRING tag:

```
<SEARCHSTATE.FROMSTRING NAME=ss VALUE= Variables.search/>
```

20.6 Coding Templates That Display Flex Assets

Are you coding templates for an online site's flex asset model? Your primary concern should be flex attributes' values. And these values are assets themselves. A flex asset (a product, for example) or flex parent asset is really an abstraction of attribute values in this context.

You use searchstates to obtain the identity of the flex assets that you want to display, filtering the assets under consideration by their attribute values. The result is an

assetset of flex or flex parent assets, and it's based on attribute values. You can display the attribute values for the assets in the assetset.

Be sure that you understand the data model of the flex family (or families) that you are using before you begin coding template elements for your flex assets. See Understanding the Asset Types and Asset Models and Designing Flex Asset Types.

Topics:

- Example Data Set for the Examples in This Section

- Examples of Assetsets with One Product (Flex Asset)

- Special Cases: Flex Attributes of Type Text, Blob, and URL

- Examples of Assetsets with Multiple Products (Flex Assets)

20.6.1 Example Data Set for the Examples in This Section

The code examples in this section start with simple assetsets and searchstates that interact with a small, example data set (product flex family in these examples). The product family data set used in these examples is as follows:

Flex Asset Type	External Name (as displayed in the WebCenter Sites interfaces)	Internal Name (as used in the WebCenter Sites database)*
flex attribute	product attribute	PAttributes
flex asset	product	Products
flex parent	product parent	ProductGroups
Always use the internal name of the asset type when you use the ASSETTYPES parameter for an ASSETSET tag.	n/a	n/a

The example products in this data set are pairs of blue jeans that have the following attributes:

Attribute	Data Type	Number of Values
sku	string	single
color	string	multiple
price	integer	single
style	text	single

There are four pairs of blue jeans, defined as follows:

sku	color	price	style
jeans-1	blue	35	wide

sku	color	price	style
jeans-2	blue,black	30	straight
jeans-3	black,green	25	straight
jeans-4	green	20	wide

20.6.2 Examples of Assetsets with One Product (Flex Asset)

The code samples in this section do the following:

- Create an assetset that contains one pair of jeans, identified by its `sku` number

- Log a dependency between the product asset and the rendered page(let)

- Get and display the value for the `price` attribute and display it

- Get and display the values for the `color` attribute and display them

- Get and display the values for both the `price` and `color` attribute with the same tag (`ASSETSET.GETMULTIPLEVALUES`)

This section includes the following topics:

- Create a Searchstate and Apply It to an Assetset

- Get the Price of the Product

- Display the Price of the Product

- Get the Colors for the Product

- Display the Colors of the Product

- Create a List Object for the ASSETSET.GETMULTIPLEVALUES tag

- Get the Value for Both Price and Color with ASSETSET.GETMULTIPLEVALUES

- Display the Value of Price and Color for the jeans-2 Product

20.6.2.1 Create a Searchstate and Apply It to an Assetset

This line of code creates an unfiltered searchstate named `ss`:

```
<SEARCHSTATE.CREATE NAME="ss"/>
```

Next, we can narrow the unfiltered searchstate named `ss` so that it finds a specific product in the sample data set, by providing the `sku` of the product:

```
<SEARCHSTATE.ADDSIMPLESTANDARDCONSTRAINT NAME="ss" TYPENAME="PAttributes"
ATTRIBUTE="sku" VALUE="jeans-2"/>
```

Now we can create an assetset named `as`, applying the searchstate named `ss` to it:

```
<ASSETSET.SETSEARCHEDASSETS NAME="as" ASSETTYPES="Products"CONSTRAINT="ss"
FIXEDLIST="true"/>
```

Since the value of the `sku` attribute is unique for each product asset, there is only one product in the assetset: the one whose `sku` value is `jeans-2`.

Because this searchstate was created by querying for a hard-coded attribute value (a sku value of `jeans-2`) we know the exact contents of the `assetset`. That is why we set the `FIXEDLIST` parameter to `true`. Now the `ASSETSET.SETSEARCHEDASSET` tag logs a compositional dependency for the product asset.

20.6.2.2 Get the Price of the Product

Next, let's extract the price of this pair of jeans:

```
<ASSETSET.GETATTRIBUTEVALUES NAME="as" ATTRIBUTE="price" TYPENAME="PAttributes"
LISTVARNAME="pricelist"/>
```

Notice that even though `price` is a single-value attribute (which means the product only has one price), the `ASSETSET.GETATTRIBUTEVALUES` tag returns the value of the `price` attribute as a list variable (`LISTVARNAME=pricelist`).

20.6.2.3 Display the Price of the Product

Now the following line of code can display the price of the `jeans-2` product:

```
Price: <ics.listget listname=pricelist fieldname=value/>
And this is the result:
Price: 30
```

20.6.2.4 Get the Colors for the Product

Next, let's determine which colors this pair of jeans is available in.

As specified above, the `color` attribute is a multiple-value attribute. Because the `ASSETSET.GETATTRIBUTEVALUES` tag works the same whether an attribute is a single-value or a multiple-value attribute, we use the tag exactly as we did for single-value `price` attribute:

```
<ASSETSET.GETATTRIBUTEVALUES NAME="as" ATTRIBUTE="color" TYPENAME="PAttributes"
LISTVARNAME="colorlist"/>
```

20.6.2.5 Display the Colors of the Product

Now the following code can display the colors for the `jeans-2` product, and, because this product can have multiple colors, the code loops through the list:

```
Colors: <LOOP LIST="colorlist">
<ics.listget listname=colorlist fieldname=value/>
 
</LOOP>
```

And this is the result:

```
Colors: black blue
```

20.6.2.6 Create a List Object for the ASSETSET.GETMULTIPLEVALUES tag

In general, you should not use the `ASSETSET.GETATTRIBUTEVALUES` tag when you want to get the value for multiple attributes.

The `ASSETSET.GETMULTIPLEVALUES` tag gets and scatters the values from multiple attributes, for all the assets in an assetset. Because the tag makes only one call to the database for all the attribute values, it performs the query more efficiently than using multiple `ASSETSET.GETATTRIBUTEVALUES` tags.

Before you can use this tag, however, you must use the LISTOBJECT tags to create a list object that describes the attributes that the ASSETSET.GETMULTIPLEVALUES tag will return. The list object needs one row for each attribute that you want to get.

This next line of code creates a list object named lo that has columns named attributetypename, attributename, and direction.

```
<LISTOBJECT.CREATE NAME="lo"
COLUMNS="attributetypename,attributename,direction"/>
```

Then, this line adds a row to the list object for each attribute, color and price:

```
<LISTOBJECT.ADDROW NAME="lo" attributetypename="PAttributes" attributename="color"
direction="none"/>
<LISTOBJECT.ADDROW NAME="lo" attributetypename="PAttributes" attributename="price"
direction="none"/>
```

The next line of code converts the list object to a list variable name lolist:

```
<LISTOBJECT.TOLIST NAME="lo" LISTVARNAME="lolist"/>
```

20.6.2.7 Get the Value for Both Price and Color with ASSETSET.GETMULTIPLEVALUES

And now we can get the values for both the price and the color attribute from our original assetset, named as:

```
<ASSETSET.GETMULTIPLEVALUES NAME="as" PREFIX="multi" LIST="lolist" BYASSET="false"/>
```

20.6.2.8 Display the Value of Price and Color for the jeans-2 Product

Now that the values are stored in the list variable (lolist), the following code can display all the values for all the attributes:

```
<LOOP LIST="lolist">
<ics.listget listname=lolist fieldname=attributename output=attrName/>
<ics.getvar name=attrName/> is
<LOOP LIST="multi:Variables.attrName">
<ics.listget listname=multi:Variables.attrNamefieldname=value/> 
</LOOP><P/>
</LOOP>
```

This code sets up a nested loop that loops through all the attributes in the lolist variable, and then loops through all the distinct attribute values for each of the attributes in the lolist list variable.

And this is the result:

```
color is blue black
price is 30
```

20.6.3 Special Cases: Flex Attributes of Type Text, Blob, and URL

To display the values held in flex attributes of type text or blob, use the methodologies described in this section.

This section includes the following topics:

- About Flex Attributes of Type Text

- About Flex Attributes of Type Blob

- Creating a BlobServer URL

- Getting and Displaying the Value of a Blob Flex Attribute

20.6.3.1 About Flex Attributes of Type Text

The `ASSETSET.GETMULTIPLEVALUES` tag does not retrieve the values for attributes of type `text`. This means that you must include a separate `ASSETSET.GETATTRIBUTEVALUES` tag for attributes of this type.

For example, if the color attribute in the sample data set used in these examples were an attribute of type `text` rather than type `string`, we could not have retrieved its values with the `ASSETSET.GETMULTIPLEVALUES` tag in the preceding examples.

20.6.3.2 About Flex Attributes of Type Blob

The `_Mungo` table for a flex asset type stores the attribute values for the flex assets of that type and the `ASSETSET` tags query the asset type's `_Mungo` table for attribute values.

Attributes of type `blob` are an exception:

- WebCenter Sites stores all the values of all the attributes of type `blob` in the `MungoBlobs` table.

- A row in the `_Mungo` table (`Products_Mungo`, for example) for an attribute of type `blob` stores only the ID of the row in the `MungoBlobs` table that holds its value. That is, the `blob` column in a `_Mungo` table is a foreign key to the `MungoBlobs` table.

This means that for an attribute of type `blob`, the `ASSETSET.GETATTRIBUTEVALUES` and `ASSETSET.GETMULTIPLEVALUES` tags return the ID of the blob attribute's value, but not the actual value.

Once the ID of the attribute's value has been identified, you can do one of two things with it:

- Use the ID to obtain a BlobServer URL.

- Use the ID to extract the actual value of the blob.

20.6.3.3 Creating a BlobServer URL

To obtain a BlobServer URL for the value of the flex attribute blob, you do the following:

- Use the `BLOBSERVICE` tags to programmatically identify the `MungoBlobs` table and the appropriate columns in it.

- Pass that information to a `RENDER.SATELLITEBLOB` tag, if you want the URL in an HTML tag, or to a `RENDER.GETBLOBURL` tag if you need only the URL without the HTML tag.

- Use the `BLOBSERVICE` tags to programmatically identify the `MungoBlobs` table, as shown in the following example. By obtaining the value with the `BLOBSERVICE` tags rather than hard coding the name of the table into your code, your code will function properly even if the table name is changed in a future version of the product.

  To illustrate the following blob examples, let's add the following attribute to the jeans products in our sample data set:

Attribute	Data Type	Number of Values
description	blob	single

- First, let's create the assetset and log the dependency between the `jeans-2` product and the rendered page:

```
<SEARCHSTATE.CREATE NAME="ss"/>
<SEARCHSTATE.ADDSIMPLESTANDARDCONSTRAINT NAME="ss" TYPENAME="PAttributes"
ATTRIBUTE="sku" VALUE="jeans-2"/>
<ASSETSET.SETSEARCHEDASSETS NAME="as" ASSETTYPES="Products"
CONSTRAINT="ss"/>
<ASSETSET.GETASSETLIST NAME="as" LISTVARNAME="aslist"/>
<RENDER.LOGDEP cid="aslist.assetid" c="aslist.assettype"/>
The next line of code gets the ID of the jeans-2 asset's description
attribute (that attribute of type blob) and stores it in a list variable
called descFile
<ASSETSET.GETATTRIBUTEVALUES NAME="as" TYPENAME="PAttributes"
ATTRIBUTE="description" LISTVARNAME="descFile"/>
```

- The next lines of code use the `BLOBSERVICE` tags to obtain the table name and column names from the WebCenter Sites table that stores the attribute values for blob attributes and store them in variables named "uTabname", "idColumn", and "uColumn":

```
<BLOBSERVICE.GETTABLENAME VARNAME="uTabname"/>
<BLOBSERVICE.GETIDCOLUMN VARNAME="idColumn"/>
<BLOBSERVICE.GETURLCOLUMN VARNAME="uColumn"/>
```

- Now we can pass the list variable named `descFile` and the `uTabname`, `idColumn`, and `uColumn` variables to a `RENDER.SATELLITEBLOB` tag, which returns a BlobServer URL in an HTML tag:

```
<RENDER.SATELLITEBLOB
BLOBTABLE="Variables.uTabname"
BLOBWHERE="descFile.value"
BLOBKEY="Variables.idColumn"
BLOBCOL="Variables.uColumn"
BLOBHEADER="application/pdf"
/> add service=a href ... download link...
```

The `RENDER.SATELLITEBLOB` tag returns a BlobServer URL in an HREF tag.

20.6.3.4 Getting and Displaying the Value of a Blob Flex Attribute

To obtain and display the contents or data in the blob flex attribute after its ID has been returned, you use a `BLOBSERVICE.READDATA` tag, which loads the file name and URL data of the blob.

- Under the same assumptions about the data set used for the preceding blob example, create the assetset, log the dependency between the `jeans-2` asset and the rendered page, and get the ID of the `description` attribute's value:

```
<SEARCHSTATE.CREATE NAME="ss"/>
<SEARCHSTATE.ADDSIMPLESTANDARDCONSTRAINT NAME="ss" TYPENAME="PAttributes"
ATTRIBUTE="sku" VALUE="jeans-2"/>
<ASSETSET.SETSEARCHEDASSETS NAME="as" ASSETTYPES="Products" CONSTRAINT="ss"/>
<ASSETSET.GETASSETLIST NAME="as" LISTVARNAME="aslist"/>
<RENDER.LOGDEP cid="aslist.assetid" c="aslist.assettype"/>
```

```
<ASSETSET.GETATTRIBUTEVALUES NAME="as" TYPENAME="PAttributes"
ATTRIBUTE="description" LISTVARNAME="descFile"/>
```

This time, get and then display the value (data) of the description attribute, using the `BLOBSERVICE.READDATA` tag:

```
<BLOBSERVICE.READDATA ID="descFile.value" LISTVARNAME="descData"/>
<ics.listget listname=descData fieldname=@urldata/>
```

20.6.4 Examples of Assetsets with Multiple Products (Flex Assets)

The code samples in this section do the following:

- Create an assetset that holds all the products (pairs of jeans) in the sample data set being used in this chapter.

- Get and display a count of the number of jeans in the assetset.

- Get and display all the values for the `color` attribute for all the pairs of jeans in the assetset.

- Get and display all the values for both the `color` and the `style` attributes for the jeans in the assetset.

- Get and display, in a table, all the attribute values for the jeans in the assetset.

- Add a search constraint that filters the assetset for the jeans whose price falls into a specific range.

- Replace the range constraint on the price attribute with a search constraint that filters the assetset for the jeans that are available in any color that begins with the letter b.

- Replace that color constraint with one that filters the assetset for the jeans that are available in either of two specific colors: blue or black.

This section includes the following topics:

- Creating a Searchstate and Apply it to an Assetset

- Displaying the Number of Assets in the Assetset

- Displaying the Colors That the Jeans Are Available In

- Displaying Both the Colors and the Styles for the Jeans in the Assetset

- Creating a Table That Displays All the Jeans and Their Attribute Values

- Searching for Jeans Based on a Range of Prices

- Searching for Jeans with a Wildcard for Color

- Searching for Jeans with Specific Colors

20.6.4.1 Creating a Searchstate and Apply it to an Assetset

This line of code creates an unfiltered searchstate named `ss`:

```
<SEARCHSTATE.CREATE NAME="ss"/>
```

When you apply the unfiltered searchstate to an assetset, you get all the flex assets of the type specified (in this case, product assets):

```
<ASSETSET.SETSEARCHEDASSETS NAME="as" CONSTRAINT="ss" ASSETTYPES="Products"/>
```

20.6.4.2 Displaying the Number of Assets in the Assetset

These lines of code return and display a count of the number of assets in the assetset, which at this point represents the entire sample catalog:

```
<ASSETSET.GETASSETCOUNT NAME="as" VARNAME="count"/>
How many products are in the catalog?
<ics.getvar name=count/>
```

And this is the result:

```
How many products are in the catalog? 4
```

20.6.4.3 Displaying the Colors That the Jeans Are Available In

The next lines of code get and display the different colors for the jeans. In other words, the distinct values of the `color` attribute:

```
<ASSETSET.GETATTRIBUTEVALUES NAME="as" ATTRIBUTE="color" TYPENAME="PAttributes"
LISTVARNAME="colors"/>
What are the possible colors for any pair of jeans?<BR/>
<LOOP LIST="colors">
<ics.listget listname=colors fieldname=value/>
</LOOP><p/>
```

And this is the result:

```
What are the possible colors for any pair of jeans?
black blue green
```

20.6.4.4 Displaying Both the Colors and the Styles for the Jeans in the Assetset

Next, let's extract and display the values for both the `color` and the `style` attribute for the jeans in the assetset. This time we use the ASSETSET.GETMULTIPLEVALUES tag.

First, however, we have to create a list object for the resultset that the ASSETSET.GETMULTIPLEVALUES tag returns. The list object needs one row for each of the attributes, as follows:

```
<LISTOBJECT.CREATE NAME="lo" COLUMNS="attributename,attributetypename,direction"/>
<LISTOBJECT.ADDROW NAME="lo" attributename="color" attributetypename="PAttributes"
direction="none"/>
<LISTOBJECT.ADDROW NAME="lo" attributename="style" attributetypename="PAttributes"
direction="none"/>
```

The next line of code converts the list object to a list variable named `lolist`:

```
<LISTOBJECT.TOLIST NAME="lo" LISTVARNAME="lolist"/>
```

Now we can extract the attributes and store them in the list variable named `lolist`:

```
<ASSETSET.GETMULTIPLEVALUES NAME="as" LIST="lolist" PREFIX="distinct"
BYASSET="false"/>
```

Notice the BYASSET parameter in the preceding line of code. Because there are multiple assets in the assetset and we want to know the distinct values for the attribute rather than all the attribute values for each asset in the assetset, BYASSET=false. This way, we get only the unique attribute values and not every single attribute value.

The next lines of code loop through the list and display the unique values for each attribute:

```
Here are all the possible colors:
<LOOP LIST="distinct:color">
<ics.listget listname=distinct:color fieldname=value/>
</LOOP><P/>

Here are all the possible styles:
<LOOP LIST="distinct:style">
<ics.listget listname=distinct:style fieldname=value/>
</LOOP><P/>
```

And this is the result:

```
Here are all the possible colors: green blue black
Here are all the possible styles: wide straight
```

20.6.4.5 Creating a Table That Displays All the Jeans and Their Attribute Values

You can also use the `ASSETSET.GETMULTIPLEVALUES` tag to obtain the attribute values that are distinct for each asset in the assetset. It creates a list of all the products and the values for their attributes that we can use to create a grid or table that displays all the products in the example catalog.

In this case, we have to do two additional things:

- Because we want the attribute values grouped by the asset that they belong to, the `BYASSET` parameter must be set to `true`.

- Because we need the IDs of the assets in this case, we have to use the `ASSETSET.GETASSETLIST` tag to obtain them.

First, this code creates a list object:

```
<LISTOBJECT.CREATE NAME="lo" COLUMNS="attributename,attributetypename,direction"/>
<LISTOBJECT.ADDROW NAME="lo" attributename="color" attributetypename="PAttributes"
direction="none"/>
<LISTOBJECT.ADDROW NAME="lo" attributename="style" attributetypename="PAttributes"
direction="none"/>
<LISTOBJECT.ADDROW NAME="lo" attributename="price" attributetypename="PAttributes"
direction="none"/>
<LISTOBJECT.ADDROW NAME="lo" attributename="sku" attributetypename="PAttributes"
direction="none"/>
<LISTOBJECT.TOLIST NAME="lo" LISTVARNAME="lolist"/>
```

Next, we can get the attribute values:

```
<ASSETSET.GETMULTIPLEVALUES NAME="as" LIST="lolist" PREFIX="grid" BYASSET="true"/>
```

And then we use the `ASSETSET.GETASSETLIST` tag.

```
<ASSETSET.GETASSETLIST NAME="as" LISTVARNAME="aslist"/>
```

It returns a list with these columns:

- assettype

- assetid

By using both lists, we can create a grid that shows all of the products and all of their attribute values:

```
<TABLE>
<LOOP LIST="aslist">
<TR>
<TD><CSVAR NAME="grid:aslist.assetid:sku.value"/></TD>
<TD><CSVAR NAME="grid:aslist.assetid:price.value"/>
    </TD>
<TD><CSVAR NAME="grid:aslist.assetid:style.value"/>
</TD>
<TD>
<IF COND="IsList.grid:aslist.assetid:color=true"><THEN>
<LOOP LIST="grid:aslist.assetid:color">
<CSVAR NAME="grid:aslist.assetid:color.value"/> 
</LOOP>
</THEN></IF>
</TD>
</TR>
</LOOP>
</TABLE>
```

And this is the result:

Table 20-1   Table That Displays All Jeans and Their Attribute Values

SKU	Price	Style	Color
jeans-1	35	wide	blue
jeans-2	30	straight	black blue
jeans-3	25	straight	black green
jeans-4	20	wide	green

20.6.4.6 Searching for Jeans Based on a Range of Prices

Up until now, we have been using the same assetset (NAME=as) that was created in the second line of code in this section. Next, let's filter the assetset by the price attribute, using a range constraint.

This line of code adds a range constraint to our original searchstate (NAME=ss) that was created in the first line of code in this section:

```
<SEARCHSTATE.ADDRANGECONSTRAINT NAME="ss" ATTRIBUTE="price" TYPENAME="PAttributes"
LOWER="0" UPPEREQUAL="30"/>
```

The range is from 0 to 30. Let's apply the modified searchstate against our assetset:

```
<ASSETSET.SETSEARCHEDASSETS NAME="as" CONSTRAINT="ss" ASSETTYPES="Products"/>
```

And check whether it worked, by obtaining and displaying a count of the jeans that are now in the assetset:

```
<ASSETSET.GETASSETCOUNT NAME="as" VARNAME="count"/>
How many jeans products are less than or equal to $30?
<ics.getvar name=count/>
```

Here's the result:

```
How many jeans products are less than or equal to $30? 3
```

20.6.4.7 Searching for Jeans with a Wildcard for Color

Now let's replace the range constraint on the `price` attribute with a search constraint that filters the assetset for the jeans that are available in any color that begins with the letter b.

First this line of code deletes the range constraint for price:

```
<SEARCHSTATE.DELETECONSTRAINT NAME="ss" ATTRIBUTE="price"/>
```

And this line of code adds a new constraint for color, using the percentage (%) character as a wildcard with the `VALUE` parameter:

```
<SEARCHSTATE.ADDSIMPLELIKECONSTRAINT NAME="ss" ATTRIBUTE="color"
TYPENAME="PAttributes" VALUE="b%"/>
```

The `VALUE="b%"` statement means any color that begins with the letter b. Lets apply the modified searchstate against our same assetset (`as`):

```
<ASSETSET.SETSEARCHEDASSETS NAME="as" CONSTRAINT="ss" ASSETTYPES="Products"/>
```

And check whether it worked by obtaining and displaying a count of the number of jeans that are in the assetset now:

```
<ASSETSET.GETASSETCOUNT NAME="as" VARNAME="count"/>
How many jeans have a color that begins with the letter b?
<ics.getvar name=count/>
```

Here's the result:

```
How many jeans have a color that begins with the letter b? 3
```

20.6.4.8 Searching for Jeans with Specific Colors

Finally, let's change the color constraint that filters the assetset for the jeans that are available in either of two specific colors: blue or black.

This line of code deletes the color constraint from the searchstate:

```
<SEARCHSTATE.DELETECONSTRAINT NAME="ss" ATTRIBUTE="color"/>
```

Next, because we want to filter based on two values for the color attribute, we have to create a list object with those values:

```
<LISTOBJECT.CREATE NAME="lo" COLUMNS="value"/>
<LISTOBJECT.ADDROW NAME="lo" value="blue"/>
<LISTOBJECT.ADDROW NAME="lo" value="black"/>
<LISTOBJECT.TOLIST NAME="lo" LISTVARNAME="colorlist"/>
```

Now we can use the list variable named `colorlist` to create the searchstate:

```
<SEARCHSTATE.ADDSTANDARDCONSTRAINT NAME="ss" ATTRIBUTE="color"
TYPENAME="PAttributes" LIST="colorlist"/>
```

The `LIST=colorlist` statement is the equivalent of the `VALUE` statement in the preceding example. It means attribute values that match any of the colors in the list named `colorlist`. Let's apply the modified searchstate to our same assetset:

```
<ASSETSET.SETSEARCHEDASSETS NAME="as" CONSTRAINT="ss" ASSETTYPES="Products"/>
```

And check whether it worked by obtaining and displaying a count of the number of jeans that are in the assetset now:

```
<ASSETSET.GETASSETCOUNT NAME="as" VARNAME="count"/>
How many products have a color that is black or blue?
<ics.getvar name=count/>
```

Here's the result:

```
How many products have a color that is black or blue? 3
```

20.7 Creating URLs for Hyperlinks

Your site may be dynamic or static, but its content changes regularly. So, you can't hard code URLs into hyperlinks. At the time of rendering, your pages must be able to determine the identity of the assets they are providing links to.

WebCenter Sites provides three tags (each with an XML and a JSP version) that you can use to create your URLs:

- For URLs for assets that are not blobs, use RENDER.GETPAGEURL tag.

- For URLs for assets that are blobs, use either the RENDER.SATELLITEBLOB tag or the RENDER.GETBLOBURL tag.

See these topics:

- RENDER.GETPAGEURL (render:getpageurl)

- RENDER.SATELLITEBLOB (render:satelliteblob)

- RENDER.GETBLOBURL (render:getbloburl)

- Using the referURL Variable

20.7.1 RENDER.GETPAGEURL (render:getpageurl)

To obtain URLs for regular assets (that is, assets that are not blobs), use the RENDER.GETPAGEURL tag.

The RENDER.GETPAGEURL tag processes arguments passed in from the element that invokes it into a URL-encoded string that it returns as a variable that you name with the OUTSTR parameter. By convention, the name of that variable is referURL.

If rendermode is set to export, it creates a static URL (unless you specify that it should be dynamic). If rendermode is set to live, it creates a dynamic URL.

For example:

```
<RENDER.GETPAGEURL PAGENAME="FiscalNews/Article/Full
cid="Variables.cid"
c="Article"
p="Variables.p"
OUTSTR="referURL"/>
```

You can now use the value in the referURL variable to create a hyperlink with an <A HREF> tag.

See the *Oracle Fusion Middleware Tag Reference for Oracle WebCenter Sites Reference*.

20.7.2 RENDER.SATELLITEBLOB (render:satelliteblob)

Binary large objects (blobs) that are stored in the WebCenter Sites database are served by the BlobServer servlet rather than the WebCenter Sites servlet. The RENDER.SATELLITEBLOB tag returns an HTML tag with a BlobServer URL in it.

This tag takes a set of arguments that define the blob and an additional set of arguments that determine how to format the blob. For example, you can use it to create an `<IMG SRC>` tag or an `<A HREF>` tag, as follows:

```
<RENDER.SATELLITEBLOB
BLOBTABLE=ImageFile
BLOBKEY=id
BLOBCOL=urlpicture
BLOBWHERE=Variables.asset:id
BLOBHEADER=Variables.asset:mimetype
SERVICE=IMG SRC
ARGS_alt=Variables.asset:alttext
ARGS_hspace=5 ARGS_vspace=5/>
```

Note that there are additional coding steps if you are creating a URL for a flex attribute of type `blob`. See About Flex Attributes of Type Blob.

Even if you are not using Satellite Server, you should still use the `RENDER.SATELLITEBLOB` tag because the tag can create a BlobServer URL in an HTML tag even when Satellite Server is not present.

See the *Oracle Fusion Middleware Tag Reference for Oracle WebCenter Sites Reference*.

20.7.3 RENDER.GETBLOBURL (render:getbloburl)

If you need a BlobServer URL only, without it being embedded in an HTML tag, use the `RENDER.GETBLOBURL` tag.

For example, the following element named `SetHTMLHeader` uses the `RENDER.GETBLOBURL` element to obtain a BlobServer URL (stored as a variable named `referURL`) that it then passes on to JavaScript code that runs on the client side to determine which browser the visitor is using. In this case, the client-side JavaScript creates the HTML tag based on the browser it discovers, so it needs the BlobServer URL without an HTML tag.

`SetHTMLHeader` is the element for a CSElement. You could examine it in two ways:

- Use the WebCenter Sites interface to search for the *path-name*/`SetHTMLHeader` CSElement and then inspect it.

- Use Explorer to open the *path-name*/`SetHTMLHeader` element.

If you are creating a URL for a flex attribute of type `blob`, there are additional coding steps. See About Flex Attributes of Type Blob.

See the *Oracle Fusion Middleware Tag Reference for Oracle WebCenter Sites Reference*.

20.7.4 Using the referURL Variable

The `RENDER.GETPAGEURL`, `RENDER.GETBLOBURL`, and `RENDER.SATELLITEBLOB` tags were introduced in the 3.6.x version of this product. Older versions of the product used elements named `GetPageURL` and `GetBlobURL` to obtain URLs; they are coded to return URLs in a variable named `referURL`.

By convention, all of the sample code in the sample sites that use the tags that replaced the `GetPageURL` and `GetBlobURL` elements use a `referURL` variable for the value of the URL.

Do not append or add any text to the value held in the `referURL` variable or any other variable returned by a `RENDER.GETPAGEURL` or `RENDER.GETBLOBURL` tag.

URLs in this kind of variable are complete (whole). If you change the URL returned by the tag, you are likely to break it.

If you have to include additional arguments in a URL, use the `RENDER.PACKARGS` tag to URL-encode them (pack them) and then pass those encoded arguments to the `RENDER.GETPAGEURL` or `RENDER.GETBLOBURL` tag with the `PACKEDARGS` parameter.

See the *Oracle Fusion Middleware Tag Reference for Oracle WebCenter Sites Reference*.

20.8 Handling Error Conditions

Can your element code check for error conditions? Decide which conditions are serious and, when necessary, code a solution or alternate action. Sometimes the solution is to write a meaningful error message. Additionally, you can also include a code that stops a broken page from being cached.

Note:

While debugging your code, you can use the `commons-logging.properties` and `loggingconfig.xml` files to enable loggers. Error and debugging messages are then written to the WebCenter Sites log file. For information about the debugging properties, see the *Oracle Fusion Middleware Property Files Reference for Oracle WebCenter Sites*.

Topics:

* Using the Errno Variable

* Ensuring that Incorrect Pages Are Not Cached

20.8.1 Using the Errno Variable

The `errno` variable, a standard WebCenter Sites variable, holds error numbers that the WebCenter Sites XML and JSP tags report. When a WebCenter Sites tag cannot successfully execute, it sets errno to the value that best describes the reason why it did not succeed. For example, an errno value of -13004 means a `CURRENCY` tag couldn't read a number because it was not in the correct currency format. For a complete list of all the errno values and their descriptions, see the error conditions section in the *Oracle Fusion Middleware Tag Reference for Oracle WebCenter Sites Reference*.

The tags that are delivered with the WebCenter Sites modules and products clear errno before they execute so you do not have to set errno to 0 when you want to check for errors from these tags. However, it is recommended that you clear error numbers before executing a tag. Here's a code example that determines whether an `ASSET.LOAD` was successful before attempting to load the child assets:

```
<ASSET.LOAD NAME="topArticle" TYPE="Article"
OBJECTID="Variables.cid"/>
<IF COND="IsError.Variables.errno=false">
<THEN>
<ASSET.CHILDREN NAME="topArticle"
LIST="listOfChildren/>
</THEN>
</IF>
```

To check the results of the tags that are delivered by WebCenter Sites, you should include code that clears the value errno before the tag whose results you want to check. For example:

```
<SETVAR NAME=errno VALUE=0/>
```

The following code sample shows an error message that you could use while you are in the process of developing your templates:

```
<IF rendermode=preview>
<THEN>
<IF COND=IsError.Variables.errno=true>
<THEN>
<FONT COLOR=#FF0000>
Error <ics.geterrno/>
while rendering <ics.getvar name=pagename/>
with asset ID <ics.getvar name=cid/>.
</FONT>
</THEN>
</IF>
</THEN>
</IF>
```

20.8.2 Ensuring that Incorrect Pages Are Not Cached

If you can determine that the output from an element is incorrect, there is probably no need for WebCenter Sites or Satellite Server to cache the page. You can stop the page that is being generated from being cached with the `ics.disablecache` tag.

Example 1: Error Condition

To continue with the first example in Using the Errno Variable, if the article asset could not be loaded, there would also be no reason to cache the page. You could add the following ELSE statement to the IF condition in that code sample:

```
<ASSET.LOAD NAME="topArticle" TYPE="Article"
OBJECTID="Variables.cid"/>
<IF COND="IsError.Variables.errno=false">
<THEN>
<ASSET.CHILDREN NAME="topArticle"
LIST="listOfChildren/>
</THEN>
<ELSE>
<ics.disablecache/>
</ELSE>
</IF>
```

Example 2: Clear the Page From Cache if the Asset's Status is VO (Basic Assets Only)

The CacheManager on the destination system regenerates all the pages and pagelets that were affected by a publishing session. Affected pages includes those whose dependent assets were deleted.

Deleted assets have their status set to VO. The ASSET.LOAD and asset:load tags do not check the status of an asset before they execute which means they can and will load a deleted asset. Typically this isn't a problem. Why? Because an asset cannot be deleted until all links to it from other assets are removed. Therefore, when the site is regenerated there are no longer any links to a page or pagelet that would display the

deleted asset. But there is no need to leave a page or pagelet that displays a deleted asset in the cache.

The following code sample stops the page from being cached if the asset cannot be loaded or if the asset's status is deleted:

```
<ASSET.LOAD TYPE="Variables.c" OBJECTID="Variables.cid" NAME="WireStoryTextArticle"/>
<!-- if the asset cannot be loaded, then flush the pagelet from cache -->
<if COND="IsError.Variables.errno=true">
<then>
<ics.disablecache/>
</then>
</if>
<ASSET.SCATTER NAME="WireStoryTextArticle" PREFIX="asset"/>
<!-- if the asset is marked as void, then flush the pagelet from cache -->
<if COND="Variables.asset:status=VO">
<then>
<ics.disablecache/>
</then>
</if>
```

Note that you do not have to include code that checks the status of flex assets. The SEARCHSTATE and searchstate tags do not return assets that have a status of VO and the ASSETSET and assetset tags do not include assets that have a status of VO in the assetsets that they create.

20.9 Encoding Page Arguments

The encodeParameter element is called to encode arguments. In the excludeParametersLst argument value field you can include the arguments you don't want to encode. Use this method as a best practice for secure coding, for any arguments passed into a template via URL or any other way.

```
<ics:callelement element="UI/Utils/encodeParameters">
<ics:argument name="excludeParametersLst"
value="<list_of_comma_seperated_args>"/>
 </ics:callelement>
```

For example:

```
<ics:callelement element="UI/Utils/encodeParameters">
<ics:argument name="excludeParametersLst" value="attributes,
displayData,browseUrl,chartParams"/>
</ics:callelement>
```

20.10 What You May Need to Know About Securing Your Site Against XSS Attacks

Have you sanitized custom and system parameters in the custom element you've written for your site?

To avoid cross site scripting (XSS) attacks, first you should identify the vulnerable parameters, and then encode or sanitize them according to their business logic.

21

Coding Templates for In-Context and Presentation Editing

Content contributors and editors look for flexibility in the way they can create and manage content. For example, adding content in Web mode instead of using forms, controlling the presentation through page and content layouts, and so on. You give them this flexibility by making attribute data type fields editable and writing appropriate code.

- Coding Templates for In-Context Content Editing

- Coding Templates for Presentation Editing

- Enabling Content Creation for Web Mode

For information on new tags used in this chapter, see the *Oracle Fusion Middleware Tag Reference for Oracle WebCenter Sites Reference*.

21.1 Coding Templates for In-Context Content Editing

You can instrument templates in such a way that content people can create and edit content in the context of their website, instead of using the standard content forms. In-context refers to the Web Mode of the Oracle WebCenter Sites: Contributor interface.

This topic modifies the HelloDetail template introduced in Creating Templates and Wrappers. See these previous topics that build on the HelloDetail template:

- Use Case 1: Building a Layout Template for Article Assets

- Use Case 2: Using Pagelet Templates

Note:

Doctype and Internet Explorer: The in-context editorial UI might not be fully functional if Internet Explorer renders a page in quirks mode. To ensure that local settings cannot affect the user interface, it is best to ensure that pages get rendered with an appropriate `doctype` value.

See the `DOCTYPE` element description in the HTML/XHTML Reference of the Microsoft Library for details: `http://msdn.microsoft.com/`

See these topics:

- Attribute Data Types

- Making String Fields Editable

- Making Text Fields Editable

21.1.1 Attribute Data Types

Asset types are defined by one or more attributes, which can be of the following types:

- **string**: A short string (normally 255 characters max)

- **text**: A long string. Typically mapped to a CLOB database type (maximum size depends on the underlying database).

- **date**: A date field

- **binary**: A BLOB attribute. Typically meant to store binary files such as image files, PDFs, etc.

- **asset**: A reference to another asset

- **number**: An integer, float, or money data type

21.1.2 Making String Fields Editable

You can make string fields editable. These fields are restricted in length (usually 255 characters) and are best suited to hold content metadata such as article headlines, author, and so on.

The following steps modify the HelloDetail template introduced in Creating Templates and Wrappers to make the article headline field editable.

You can use the `ics:listget` tag to print the value of the headline field (which is stored in the value column of the `article:headline` list).

```
<h1>
<ics:listget
  listname="article:headline"
  fieldname="value" />
</h1>
```

1. To make this field editable in-context, add the following `taglib` directive:

   ```
   <@ taglib prefix="insite" uri="futuretense_cs/insite.tld" %>
   ```

   and replace the previous code with the code snippet below:

   ```
   <h1>
   <insite:edit
     field="headline"
     list="article:headline" column="value"
     assetid='<%=ics.GetVar("cid")%>'
     assettype='<%=ics.GetVar("c")%>' />
   </h1>
   ```

   where:

   - The field parameter designates which field of the asset is edited.

- The `list` and `column` parameters designate where the current field value is stored.

- The `assetid` and `assettype` parameters designate the edited asset.

The syntax can be simplified as follows:

```
<h1>
<insite:edit
  field="headline"
  list="article:headline"
  column="value" />
</h1>
```

Because the edited asset is designated by the asset type (c) and asset id (cid) variables, WebCenter Sites performs the retrieval of these values directly from the ICS scope.

Examining any asset in Web Mode of the Contributor interface, using HelloArticleLayout will not show any noticeable differences from the previous display. In fact, when in preview or inspect view, the `insite:edit` tag behaves exactly like the `ics:listget` tag we just replaced.

2. Selecting **Edit View** makes all editable areas active; in this case, the headline field as shown in the figure below.

Figure 21-1   Sample Web Mode Page with Editable Area Active

3. Editorial users are now able to edit the headline field in the context of the article detail web page and click **Save** to make this change permanent. This action is equivalent to going to the article asset form, editing the **Headline** field of the article, as shown in the figure below, and then clicking **Save**.

Figure 21-2   Sample Article Asset Form (Content Tab)

21.1.2.1 Variants of the <insite:edit/> Tag

The value of an edited field is stored in a list. This is how the assetset tags work; they query the database and return attribute values in a list object whether attributes are single-valued or multivalued.

In some cases, the value might be made available in a variable. The following variant of `insite:edit` can then be used:

```
<h1>
<insite:edit
  variable="headlineVar"
  field="headline" />
</h1>
```

In a case where the field value is not available in a variable or a list, it is possible to specify the value directly using the `value` attribute in the `insite:edit` tag.

```
<%String headline = getHeadlineValueFromSomewhere(); %>
<insite:edit
  value="<%=headline%>"
  field="headline" />
```

The `insite:edit` tag also supports the `property` and `ssvariable` attributes, in case the field value is available in, respectively, a WebCenter Sites property file or a session variable. These variants are rarely used.

21.1.3 Making Text Fields Editable

You can make text fields editable. This topic builds on the HelloDetail template introduced in Creating Templates and Wrappers.

In a similar way as for string fields, you can replace the `render:stream` tag in the HelloDetail template with the `insite:edit` tag. See Making String Fields Editable for an example of working with string fields in the HelloDetail template:

Previously, this was the `render:stream` tag:

```
<render:stream
  list="bodyList"
  column="value" />
```

To make this field editable:

1. Change the `render:stream` tag: to an `insite:edit` tag.

   In this example, the body is defined as the field and ckeditor as the editor type.

   ```
   <insite:edit
     list="bodyList"
     column="value"
     field="body"
     editor="ckeditor" />
   ```

2. Clicking on the body text displays the CKEditor widget as shown in the figure below.

   Figure 21-3   Sample CKEditor Widget
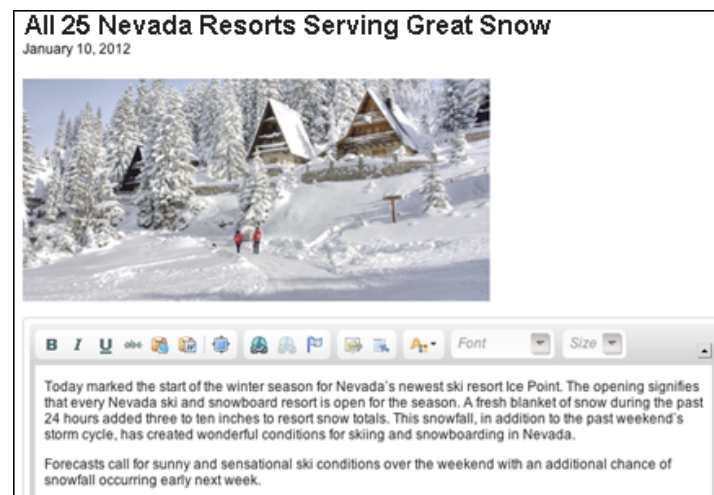
3. You can pass additional arguments to CKEditor, as follows:

   ```
   <insite:edit
     list="bodyList"
     column="value"
     field="body"
     editor="ckeditor"
     params="{width: '500px', height: '350px',
             toolbar: 'MyToolbar'}" />
   ```

   See Configuring CKEditor.

21.1.4 Making Date Fields Editable

You can make date fields editable. This information builds on the HelloDetail template introduced in Creating Templates and Wrappers.

Date fields are made editable in-context using the same `insite:edit` tag used for string or text fields. However, when dates have to be formatted, a few extra steps are required. The value of date fields when initially retrieved from the database is in JDBC format, which is, `2012-01-01 00:00:00.0`. This format is unsuitable for rendering on a website where dates are generally rendered as a readable string, such as `January 1, 2012`.

In addition, the date is interpreted in the server's time zone, which we can display in a different time zone. We first get the time zone ID by using the `java.util` TimeZone

API. We then format the date by using the `long` date format (which is a predefined format).

Note:

For more about predefined date formats, see:

http://docs.oracle.com/javase/tutorial/i18n/format/
dateFormat.html

If you choose to format the date, you can use one of the date formatting APIs, described in the next section. Depending on your decision, the resulting date and calendar widget will look like one of the samples in the figure below.

Figure 21-4   Date and Calendar Widget
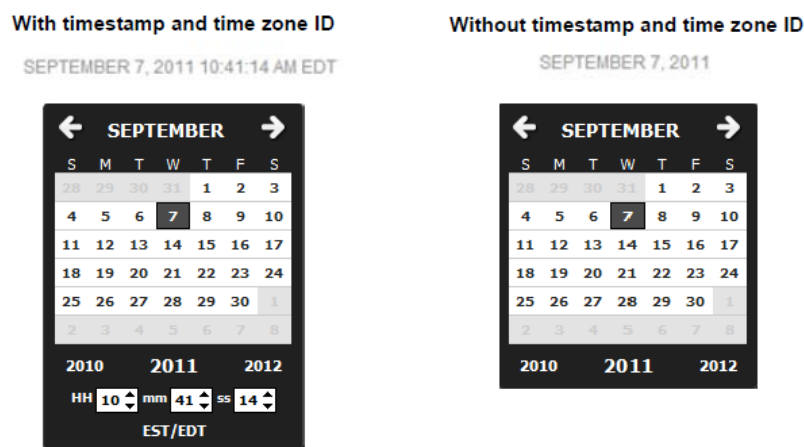
21.1.4.1 Date Formatting APIs

The HelloDetail template uses the `formattedDate` variable to display the date. See Use Case 1: Building a Layout Template for Article Assets.

```
<span class="date">
  <ics:getvar name="formattedDate" />
</span>
```

The `formattedDate` variable must be set by using one of the date formatting APIs: `fmt:formatDate` or the WebCenter Sites `dateformat` API.

The `formattedDate` can be rendered with a timestamp if you add the time style parameter. It can also be rendered with a specific time zone if we add a parameter for time zone. If the time zone parameter is omitted, the date is interpreted and rendered in the server's time zone. Following is a description of the APIs:

- The `fmt:formatDate` (which supports EL expression) is one such API, which takes the date in `java.util.Date()` in String format and takes the `timeZone` parameter as shown below:

  ```
  <fmt:formatDate value="${asset.postDate}" dateStyle="long"
    type="both" timeStyle="long" var="formattedDate"
    timeZone="US/Eastern" />
  ```

- The WebCenter Sites `dateformat` API has an additional parameter named `timezoneid` and may be used in place of `fmt:formatDate`:

```
<dateformat:create name="dateFormat" datestyle="long"
  timestyle="long" timezoneid="US/Eastern" />
```

If timestamp is needed, we use `dateformat:getdatetime`:

```
<dateformat:getdatetime name="dateFormat" value='<%=postDate%>'
  valuetype="jdbc" varname="formattedDate"/>
```

If timestamp is not needed, we use `dateformat:getdate`:

```
<dateformat:getdate name="dateFormat" value='<%=postDate%>'
  valuetype="jdbc" varname="formattedDate"/>
```

The WebCenter Sites `dateformat` API takes `millis` or `jdbc` in the `valuetype` argument.

21.1.4.2 Enabling Date Fields for Editing in Web Mode

A formatted date can be used in the `insite:edit` tag. Do the following steps to enable date fields for editing in Web Mode:

1. Because the `insite:edit` tag passes the formatted value, an appropriate `formatLength` parameter is required in the `params` argument. In our example, `formatLength` is set to set to `long`, as shown below. In addition, if we choose to display the timestamps and time zone in date fields, we would set two additional parameters; `timePicker:true` and `timeZoneID:'US/Eastern'`, also shown below:

```
<span class="date">
  <insite:edit
  field="postDate"
  value="formattedDate"
  params="{constraints:{formatLength: 'long'},
    timePicker:true, timeZoneID:'US/Eastern'}" />
</span>
```

> **Note:**
>
> The date widget in edit mode will function correctly only if we pass a properly formatted date. The `datestyle` and `timestyle` arguments in `dateformat:create` or `fmt:formatdate` APIs must be consistent with the `formatLength` params argument in the `insite:edit` tag. In our example, the datestyle and timestyle arguments in the date formatting API must all be `long`. If the `formattedDate` is constructed using the time zone parameter, then the same time zone ID must be used in the `insite:edit` tag. If the `formattedDate` is constructed with timestamp using `dateformat:getdatetime`, then the `timePicker:true` parameter must be set in the `insite:edit` tag. And if it is constructed without timestamp using `dateformat:getdate`, then there is no need to set the `timePicker:true` parameter.

2. The date will now be rendered according to how we specified its format:

   - With timestamp and `timeZoneID:'US/Eastern'`:

     SEPTEMBER 7, 2011 10:41:14 AM EDT

- Without timestamp and `timeZoneID`::

  SEPTEMBER 7, 2011

3. The date field is now editable. Click the date field while viewing an article in the Contributor interface in the Web Mode / Edit View. One of the calendar popup widgets opens.

The date format is passed to the `insite:edit` tag using the `params` attribute. The `params` attribute is used to pass extra configuration settings to the Dojo widgets, formatted as a JSON string. In this case:

```
{constraints: {formatLength: 'long'}, timePicker : true,
  timeZoneID:'US/Eastern' }
```

For details on available widget settings, refer to the Dojo documentation at `http://dojotoolkit.org/`.

21.1.5 Making Binary Fields Editable

Binary fields are typically database BLOB or CLOB fields. Binary fields typically store images, or downloadable documents. By default, the `insite:edit` tag will make binary fields editable through a file upload component.

> **Note:**
>
> This section also applies to WebCenter Sites URL columns. That is, URL columns storing a reference to a file which is stored on the file system.

The HelloDetail template renders the **largeThumbnail** field of the related AVIImage asset.

To make this field editable in-context:

1. Retrieve the value of this field, which contains a BLOB ID (not the actual BLOB value):

```
<assetset:setasset
  name="image"
  type="AVIImage"
  id='<%=ics.GetVar("imageId")%>' />
<assetset:getattributevalues
  name="image"
  attribute="largeThumbnail"
  typename="ContentAttribute"
  listvarname="largeThumbnail" />
```

2. Generate a URL to the `largeThumbnail` field image:

```
<render:getbloburl
  outstr="imageURL"
  c="AVIImage"
  cid='<%=ics.GetVar("imageId")%>'
  field="largeThumbnail"  />
```

3. Insert the `<img>` tag between the opening and closing `insite:edit` tag:

```
<insite:edit
  field="largeThumbnail"
```

```
        assetid='<%=ics.GetVar("imageId")%>'
        assettype="AVIImage"
        list="largeThumbnail"
        column="value" >

            <img class="photo left" src='<%=ics.GetVar("imageURL")%>' />
    </insite:edit>
```

When hovering over the image, content contributors are now shown a tooltip, giving access to the upload component, and allowed to clear the largeThumbnail field.

> **Note:**
>
> The HelloDetail template allows contributors to edit two distinct assets on the same page (the headline, date, body, relatedImage fields of the AVIArticle asset, and the **largeThumbnail** field of the related AVIImage asset). Generally, it is possible to render multiple assets on the same page, and make them editable simultaneously.

21.1.6 Making Asset Fields Editable

Asset fields store references to other assets. You can make asset fields editable.

This procedure builds on the HelloArticleLayout template created in Use Case 1: Building a Layout Template for Article Assets of Creating Templates and Wrappers.

You can enhance the HelloArticleLayout template and populate the side bar by rendering the article assets related to our article through the relatedStories field.

> **Note:**
>
> Although relatedStories is a multivalued field, it is treated in this section as a single-valued field. That is, the field holds only one related article content asset. Code samples shown in this section are therefore applicable to any single-valued field. For information about handling multivalued fields, see Multivalued Fields.

1. Create a pagelet template called HelloSideBar, applicable to the AVIArticle asset type and Article subtype, which renders the associated article asset using the avisports Summary/SideBar article template using the following code:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"%>
<%@ taglib prefix="assetset" uri="futuretense_cs/assetset.tld"%>
<%@ taglib prefix="ics" uri="futuretense_cs/ics.tld"%>
<%@ taglib prefix="render" uri="futuretense_cs/render.tld"%>


<cs:ftcs>

<render:logdep
  c="Template"
  cid='<%=ics.GetVar("tid")%>' />

<assetset:setasset
  name="article"
  type="AVIArticle"
```

```
       id='<%=ics.GetVar("cid")%>' />

<assetset:getattributevalues
  name="article"
  attribute="relatedStories"
  listvarname="relatedStories"
  typename="ContentAttribute" />

<%-- we are getting the first item in the list --%>
<ics:listget
  listname="relatedStories"
  fieldname="value"
  output="articleId" />

<render:calltemplate
  tname="Summary/SideBar"
  c="AVIArticle"
  cid='<%=ics.GetVar("articleId")%>' />

</cs:ftcs>
```

2. Modify the HelloArticleLayout template to invoke the `HelloSideBar` pagelet template inside the `side-bar` div element:

```
<div class="side-bar">
    <render:calltemplate tname="HelloSideBar" args="c,cid" />
</div>
```

Assuming that `relatedStories` contain one asset reference (Cold Snap Back on the Scene in the next figure), the related article is now rendered in the side bar using the Summary/SideBar template, as dictated by the template code.

Figure 21-5   Sample Related Stories Reference

The next figure shows how the related story is rendered.

Figure 21-6   Related Story Rendered in Sidebar

Previously, the `render:calltemplate` tag was used in the `HelloSideBar` pagelet:

```
<render:calltemplate
  tname="Summary/SideBar"
```

```
      c="Article"
      cid='<%=ics.GetVar("articleId")>' />
```

3. The `relatedStories` asset field can be made editable in-context, by turning the area occupied by the asset into a drop target. That is, an area which will accept assets dragged from other parts of the UI (such as the search result pane or the content tree). To do this, add the following `taglib` directive:

```
<%@ taglib prefix="insite" uri="futuretense_cs/insite.tld" %>
```

and replace the `render:calltemplate` tag in the HelloSideBar pagelet with the `insite:calltemplate` tag:

```
<insite:calltemplate
  tname="Summary/SideBar"
  c='AVIArticle'
  cid='<%=ics.GetVar("articleId")%>'
  field="relatedStories"
  assetid='<%=ics.GetVar("cid")%>'
  assettype='<%=ics.GetVar("c")%>' />
...
```

where:

- The `tname` (template), `c` (asset type) and `cid` (asset id) variables behave exactly as in the `render:calltemplate` tag and have the same meaning.

- The `assetid` and `assetype` variables designate the asset being edited (in our example, an article asset).

- The `field` variable designates which field of the asset is being edited (in our example, `relatedStories`).

The syntax can be simplified as follows:

```
<insite:calltemplate
  tname="Summary/SideBar"
  c='AVIArticle'
  cid='<%=ics.GetVar("articleId")%>'
  field="relatedStories" />
```

This simplified structure can be used because the edited asset is the asset designated by the `c` and `cid` variables. Thus, WebCenter Sites retrieves the variable values by looking them up directly in the ICS context.

4. If the field is initially empty (in which case the `articleId` variable is null), viewing the asset in Web Mode of the Contributor interface shows an empty content-editable slot (provides a droppable zone for the user).

5. It is now possible to drag and drop an article asset to the content-editable slot, by selecting an asset in the content tree, or from the docked search panel.

21.1.6.1 Editing an Association

When an asset association is used instead of a flex attribute of data type `asset`, the `field` attribute needs to be specified, as follows:

```
Association-named:<associationName>
```

For example, for an association called `topStory` the code would be:

```
<insite:calltemplate
  field="Association-named:topStory"
  ...
/>
```

21.1.6.2 Editing a Parent Asset

It is also possible to edit a flex asset's parent asset. The syntax for the `field` attribute is as follows:

```
Group_<parentDefinitionName>
```

For example, in the case of avisports, article assets have a `Category` parent definition:

```
<insite:calltemplate
  field="Group_Category"
  ...
/>
```

21.1.7 Number Fields

When dealing with number attributes (such as, integer, double, and money), raw values are retrieved from the database, and are typically formatted according to the current locale.

For example, assuming that `price` is a money attribute, a JSP reading the attribute value and rendering it as a formatted string could be written as follows:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld" %>
<%@ taglib prefix="assetset" uri="futuretense_cs/assetset.tld" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>

<cs:ftcs>

<assetset:setasset
  name="theAsset"
  type='<%=ics.GetVar("c")%>'
  id='<%=ics.GetVar("cid")%>' />

<assetset:getattributevalues
  name="theAsset"
  attribute="price"
  listvarname="pricelist"
  typename="ContentAttribute" />

<ics:listget
  listname="pricelist"
  fieldname="value"
  output="price" />

<fmt:formatNumber
  type="currency"
  value='<%=ics.GetVar("price")%>'
  var="formattedValue"
  currencySymbol="&eur;" />

The price is: ${formattedValue}
</cs:ftcs>
```

Assuming that the raw value is `123456`, it would be rendered as €123,456 (en_US locale). The value is made editable using the `insite:edit` tag as follows:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld" %>
<%@ taglib prefix="assetset" uri="futuretense_cs/assetset.tld" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>

<cs:ftcs>

<assetset:setasset
  name="theAsset"
  type='<%=ics.GetVar("c")%>'
  id=''<%=ics.GetVar("cid")%> />

<assetset:getattributevalues
  name="theAsset"
  attribute="price"
  listvarname="pricelist"
  typename="ContentAttribute" />

<ics:listget
  listname="pricelist"
  fieldname="value"
  output="price" />

<fmt:formatNumber
  type="currency"
  value='<%=ics.GetVar("price")%>'
  var="formattedValue"
  currencySymbol="&eur;" />

The price is: <insite:edit field="price"
                value="${formattedValue}"
                params="{currency: 'EUR'}" />
</cs:ftcs>
```

Note that the editing widget is passed the formatted value (containing the currency symbol). For this reason, the currency ISO code is specified using the `params` field.

For more configuration options, refer to the Dojo documentation at http://dojotoolkit.org/.

21.1.8 Multivalued Fields

For multivalued fields, beyond editing the existing values, you need to use editors for tasks such as these:

- Adding a new value

- Removing an existing value

- Reordering existing values

For this reason, multivalued text fields and multivalued asset fields have to be treated specifically.

21.1.8.1 Example 1: Editing Multivalued Text Fields

1. Page assets of the AVIHome subtype have a multivalued text field (see attribute value `teaserText` in the `assetset:getattributevalues` tag below). Create a test layout template, rendering only the value of this field.

   ```
   <%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"%>
   <%@ taglib prefix="ics" uri="futuretense_cs/ics.tld"%>
   ```

```
<%@ taglib prefix="assetset" uri="futuretense_cs/assetset.tld"%>
<%@ taglib prefix="render" uri="futuretense_cs/render.tld"%>


<cs:ftcs>

<render:logdep
  cid='<%=ics.GetVar("tid")%>'
  c="Template" />

<assetset:setasset
  name="page"
  type="Page"
  id='<%=ics.GetVar("cid")%>' />

<assetset:getattributevalues
  name="page"
  attribute="teaserText"
  listvarname="teaserList"
  typename="PageAttribute" />

<div style="width: 500px; font-size: small">
  <ics:listloop listname="teaserList">
    <render:stream list="teaserList" column="value" />
  </ics:listloop>
</div>
</cs:ftcs>
```

This figure shows a sample text layout template:

Figure 21-7   Sample Text Layout Template
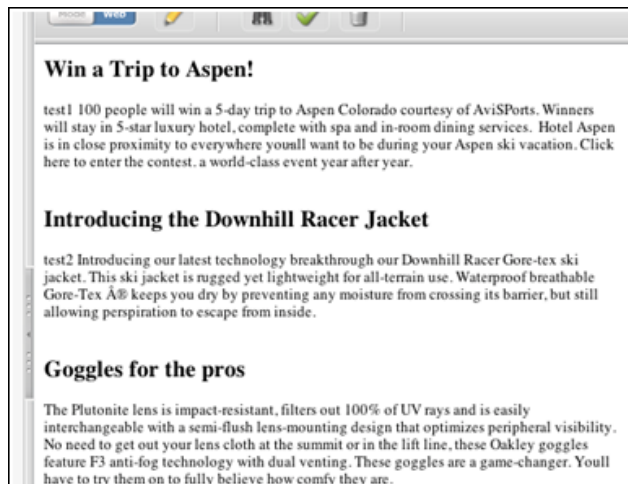
2. Modify this template to make the multivalued `teaserText` field editable.

Add the `insite` taglib directive:

```
<%@ taglib prefix="insite" uri="futuretense_cs/insite.tld" %>
```

and replace the original code:

```
<ics:listloop listname="teaserList">
  <render:stream list="teaserList" column="value" />
</ics:listloop>
```

with the following code:

```
<ics:listloop listname="teaserList">
  <ics:listget
    listname="teaserList"
    fieldname="#curRow"
    output="currentRowNb" />
  <insite:edit
    assetid='<%=ics.GetVar("cid")%>'
    assettype='<%=ics.GetVar("c")%>'
    field="teaserText"
    list="teaserList"
    column="value"
    index='<%=ics.GetVar("currentRowNb")%>'
    editor="ckeditor" />
</ics:listloop>
```

The syntax for the `insite:edit` tag can be simplified as follows:

```
<ics:listloop listname="teaserList">
  <ics:listget
    listname="teaserList"
    fieldname="#curRow"
    output="currentRowNb" />
  <insite:edit
    field="teaserText"
    list="teaserList"
    column="value"
    index='<%=ics.GetVar("currentRowNb")%>'
    editor="ckeditor" />
</ics:listloop>
```

3. In the Contributor interface, change to **Web Mode / Edit View**. You are now able to edit each of the existing values using CKEditor.

The only noticeable difference with a single-valued field, is that we only added an `index` attribute which notifies WebCenter Sites the index of the value being edited.

At this point we are only able to edit existing values. See the next section on how to add, remove, or reorder existing values.

21.1.8.2 Example 2: Modifying Multivalued Text Fields

This example builds on Example 1: Editing Multivalued Text Fields and includes adding, removing, and reordering values.

1. Modify the previous code sample as follows:

```
<insite:list
  field="teaserText"
  editor="ckeditor"
  assetid='<%=ics.GetVar("cid")%>'
  assettype='<%=ics.GetVar("c")%>'>

    <ics:listloop listname="teaserList">
      <insite:edit list="teaserList" column="value" />
    </ics:listloop>
</insite:list>
```

The `insite:edit` tag is now nested inside an `insite:list` tag. The nested `insite:edit` tags do not specify the `field`, `assetid`, `assettype` or `editor` attributes since they are specified at the parent tag level, and each nested

insite:edit tag is, by default, inheriting those values when not locally specified. This is also applicable to the params attribute of the insite:edit tag.

The index attribute is no longer required. In this case, the insite:list tag is keeping track of the current index. The tag assumes that the order in which the insite:edit tag appear matches the order of the multivalued field, that is, the first insite:edit tag edits value #1, and so on. If that is not the case, the index has to be specified. See Specifying a Different Ordering.

Like the insite:edit tag, the syntax for the insite:list tag can be simplified if the asset being edited is the asset designated by the c (asset type) and cid (asset id) variables; in which case the assetid and assettype attributes can be omitted.

```
<insite:list field="teaserText" editor="ckeditor" >
  <ics:listloop listname="teaserList">
    <insite:edit list="teaserList" column="value" />
  </ics:listloop>
</insite:list>
```

2. Now that the insite:list tag was added, a toolbar is displayed whenever the mouse pointer hovers over the area showing the field values.

When you click that area, a popup gets rendered, allowing you to add, edit, remove, or reorder field values.

This topic builds on Example 1: Editing Multivalued Text Fields and Example 2: Modifying Multivalued Text Fields.

In the case of asset reference fields, we will use the insite:slotlist tag instead of the insite:list tag. In this case, rather than nested insite:edit tags, we will have nested insite:calltemplate tags.

Instead of rendering a single related article, this example goes through the whole list and renders each of them.

3. Modify the previous code sample as follows (for the time being, this code omits any in-context editing capabilities):

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"%>
<%@ taglib prefix="assetset" uri="futuretense_cs/assetset.tld"%>
<%@ taglib prefix="ics" uri="futuretense_cs/ics.tld"%>
<%@ taglib prefix="render" uri="futuretense_cs/render.tld"%>

<cs:ftcs>

<assetset:setasset
  name="article"
  type="AVIArticle"
  id='<%=ics.GetVar("cid")%>' />

<assetset:getattributevalues
  name="article"
  attribute="relatedStories"
  listvarname="relatedStories"
  typename="ContentAttribute" />

<ics:listloop listname="relatedStories">
  <ics:listget
    listname="relatedStories"
    fieldname="value"
    output="articleId" />
```

```
<render:calltemplate
   tname="Summary/SideBar"
   c="AVIArticle"
   cid='<%=ics.GetVar("articleId")%>' />
</ics:listloop>
</cs:ftcs>
```

This figure shows the field with three values.

Figure 21-8   Related Stories with Three Examples
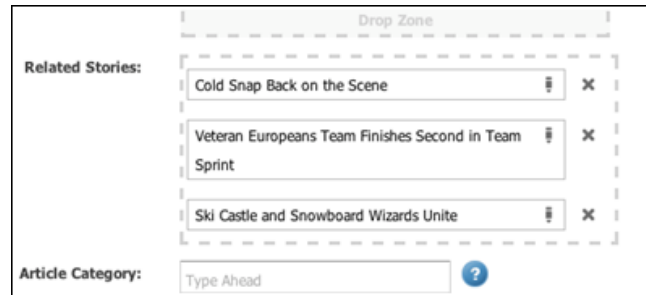
The SideBar will now show those three articles.

Figure 21-9   Related Stories Displayed in SideBar
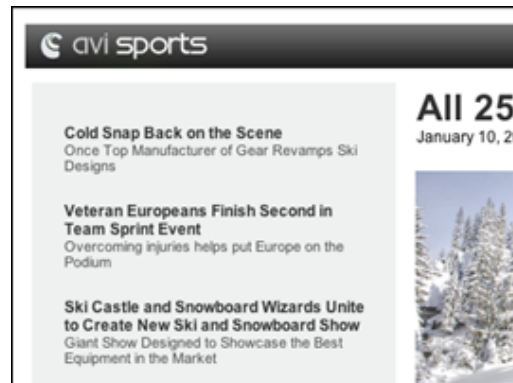
4. To make the article list editable, make the following changes:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"%>
<%@ taglib prefix="assetset" uri="futuretense_cs/ assetset.tld"%>
<%@ taglib prefix="ics" uri="futuretense_cs/ics.tld"%>
<%@ taglib prefix="render" uri="futuretense_cs/render.tld"%>
<%@ taglib prefix="insite" uri="futuretense_cs/insite.tld"%>

<cs:ftcs>
<render:logdep c="Template" cid='<%=ics.GetVar("tid")%>' />

<assetset:setasset
  name="article"
  type="AVIArticle"
  id='<%=ics.GetVar("cid")%>' />

<assetset:getattributevalues
  name="article"
  attribute="relatedStories"
  listvarname="relatedStories"
  typename="ContentAttribute" />
```

```
<insite:slotlist field="relatedStories">
  <ics:listloop listname="relatedStories">
    <ics:listget
      listname="relatedStories"
      fieldname="value"
      output="articleId" />
    <insite:calltemplate
      tname="Summary/SideBar"
      c="AVIArticle"
      cid='<%=ics.GetVar("articleId")%>' />
  </ics:listloop>
</insite:slotlist>
</cs:ftcs>
```

Every item of the list becomes a content-editable slot (droppable zone):

In addition, since the `insite:slotlist` tag was used in the example, when hovering over the `relatedStories` area, you see a toolbar similar to the previous example.

21.1.8.3 Specifying a Different Ordering

In the previous example, the articles are rendered in a sequence, the first nested `insite:calltemplate` tag renders article #1, the second `insite:calltemplate` tag renders `article #2`, and so on:

```
<div class="top-stories">
  <div>article #1</div>
  <div>article #2</div>
  <div>article #3</div>
  <div>...</div>
</ul>
```

Your structure might be different depending on the structure of the HTML markup used in your site. For example, you might want articles to be displayed in a column layout such as this:

```
article #1  article #2
article #3  article #4
etc.
```

The underlying HTML markup might be similar to this:

```
<div class="left-column">
  <div>article #1</div>
  <div>article #3</div>
  ...
</div>
<div class="right-column">
  <div>article #2</div>
  <div>article #4</div>
  ...
</div>
```

In this example, articles are rendered in an order that doesn't not match the ordering of the field (#1, #3, #5, then #2, #4, #6, and so on.). If this is the case, WebCenter Sites needs to be aware of the ordering. To do that, you have to explicitly indicate the index of the list item being edited, by using the `index` attribute as shown in Multivalued Fields.

21.1.8.4 Editing Mode and Caching

Caching is not disabled when you are working in **Web Mode / Edit View**. Therefore, templates that are configured to be cached will still be cached when rendered in Edit View.

When you are working with assets, WebCenter Sites automatically handles cache flushing provided that the correct dependencies are logged. For example, modifying the definition of a particular attribute (for example, changing the allowed types of an asset reference field) automatically flushes all pagelets whose rendering depends on this particular attribute.

However, the page cache has to be manually flushed in these cases:

- Modifying the definition of an association field

- Removing a role from WebCenter Sites, if this role is directly referenced from an `insite:calltemplate` tag, through the `roles` attribute

- Removing asset types or subtypes from WebCenter Sites, if this asset type or subtype is referenced from an `insite:calltemplate` tag through the `clegal` attribute

21.2 Coding Templates for Presentation Editing

Content contributors like to present their content in different layouts to appeal the site visitors. So, you can let them control the page and content presentation. For example, let them choose which page layout should render an entire web page, which content layout should render an asset in a part of a page, and which arguments should be sent to a pagelet template.

The following information covers context and how to make a presentation change local (that is, visible only on a given web page), or global (that is, changing the presentation on one page can propagate the same change to multiple pages on the site).

This section builds on the HelloDetail template introduced in Creating Templates and Wrappers and the examples in Coding Templates for In-Context Content Editing.

See these topics:

- Selecting a Different Layout for the Entire Web Page

- Selecting a Different Layout for a Page Fragment

- Editing Presentation and Content Simultaneously

- Understanding the Context System Variable

- Using Slots with CSElement and SiteEntry Assets

- Constraining Asset Types

- Preventing CSS and JavaScript Conflicts

21.2.1 Selecting a Different Layout for the Entire Web Page

Editorial users can assign a layout template to assets in two ways:

- Directly modify the value of the `template` field in Form Mode of the Contributor interface.

- Select the Change Layout functionality from either the toolbar or menu bar.

Figure 21-10   Change Page Layout Dialog
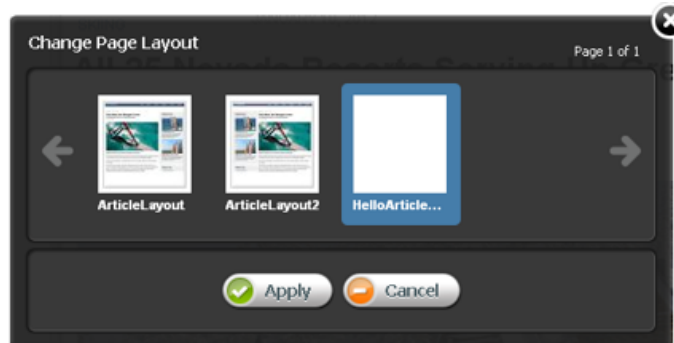
Selecting a layout template for an asset will make this template the default choice when working with an asset in Web Mode of the Contributor interface, or when previewing the asset.

To enable content contributors to control the page layout used to render a given asset on a live site, the value of the template field should be looked up and used to calculate asset hyperlinks, as in the following example.

```
<asset:list
  type='<%=ics.GetVar("c")%>'
  list="asset"
  field1="id"
  value1='<%=ics.GetVar("cid")%>' />

<ics:listget
  listname="asset"
  fieldname="template"
  output="template" />

<render:gettemplateurl
  outstr="pageURL"
  tname='<%=ics.GetVar("template")%>'
  args="c,cid" />
```

21.2.2 Selecting a Different Layout for a Page Fragment

This information builds on the HelloArticleLayout template used in Coding Templates for In-Context Content Editing. For information about working with fragments, see Developer's Samples Website in Website Development the MVC Framework and APIs.

Previously, in the HelloArticleLayout layout template, the main area of the article page was rendered using the HelloDetail template as shown in this code snippet:

```
<div id="container">
  <div class="content">
    <render:calltemplate tname="HelloDetail" args="c,cid" />
  </div>
  <div class="side-bar">
    ...
  </div>
</div>
```

This template renders output as shown in the figure.

Figure 21-11 Article Page Rendered Using Template

> **Note:**
>
> The syntax used in the code sample:
>
> ```
> <render:calltemplate tname="HelloDetail" args="c,cid" />
> ```
>
> is a shortcut for (and is strictly equivalent to):
>
> ```
> <render:calltemplate  tname="HelloDetail"  c='<%=ics.GetVar("c")%>'
> cid='<%=ics.GetVar("cid")%>' />
> ```

The avisports sample site also provides a Detail template for article assets. The Detail pagelet template is functionally equivalent to the HelloDetail template, in that it provides a detailed view of the article, and is meant to be rendered in the main area of the page, as shown in the figure.

Figure 21-12   Rendering with Detail Pagelet Template

21.2.2.1 Defining a Slot for Presentation Editing

To allow non-technical users to choose which presentation (that is, which pagelet template) to use to render a particular article page, you must define a slot.

The previous code in the HelloArticleLayout layout template rendered the main area of the article page using the HelloDetail template:

```
<div id="container">
  <div class="content">
    <render:calltemplate
      tname="HelloDetail"
      args="c,cid" />
  </div>
  <div class=side-bar>
    ...
  </div>
</div>
...
```

To define a slot:

1. Add the `insite` taglib directory.

   ```
   <%@ taglib prefix="insite" uri="futuretense_cs/insite.tld" %>
   ```

2. Replace the previous `render:calltemplate` tag with an `insite:calltemplate` tag and add the `slotname` and `variant` attributes:

   ```
   <div id="container">
     <div class="content">
       <insite:calltemplate
         slotname="HelloSlot"

         tname="HelloDetail"
         variant="HelloDetail|Detail"
         args="c,cid" />
     </div>
   ```

```
  <div class=side-bar>
    ...
  </div>
</div>
...
```

By adding the `slotname` attribute, you are defining a slot, called in the example `HelloSlot`. The `slotname` attribute value can be any string, but it must be unique across all templates of a given site. See Understanding the Context System Variable.

3. Run this template again.

Initially there are no noticeable changes. The article is still rendered using the HelloDetail template, as directed by the `tname` attribute.

4. Switch to **Edit View** and hover over the main `div` section. The changes are as follows:

- The page fragment inside the main div is now marked with a blue overlay.

- The slot name is indicated in the top right corner.

- Clicking the blue overlay also brings up a toolbar.

5. Click the **Change Content Layout** icon.

Figure 21-13   Change Content Layout Icon

The Change Content Layout Dialog opens.

Using the **Change Content Layout** option enables you to select a different content layout to render the asset in the main area of the page.

6. The template picker shows the HelloDetail and Detail pagelet templates, specified by the `variant` attribute. Select the Detail template and click **Apply** to render the web page.

Figure 21-14   Output Page with Change Content Layout Dialog
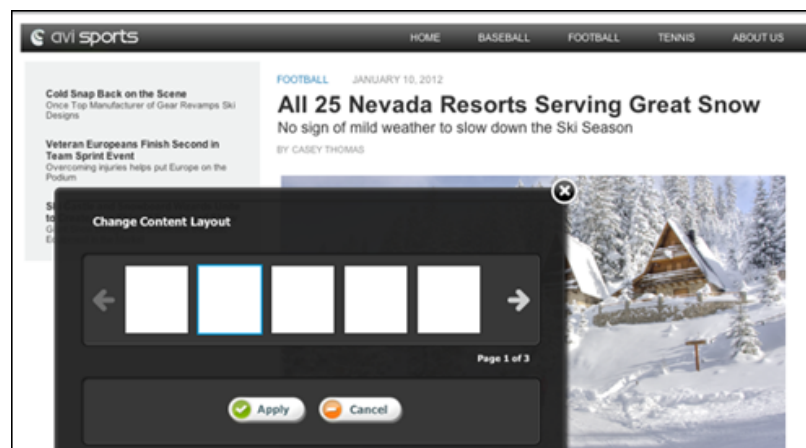
The slot defined in Figure 21-14 is only meant for presentation editing and is not a content-editable slot (not a droppable zone).

The reasons for this behavior follows:

- The `insite:calltemplate` tag does not define any `field` attribute. That is, the content of the slot is not the value of an asset reference field.

  The tag is explicitly providing values for `c` (asset type) and `cid` (asset id). In this particular case, we want the article asset specified by the incoming `c` and `cid` request parameters to be rendered in this location.

  In addition, if the `variant` attribute had been omitted, the slot layout would not have been editable, since the `insite:calltemplate` tag specifies an explicit value for the `tname` attribute (which acts as a default template for all assets dropped in this slot).

Note:

The `variant` attribute can contain any regular expression. For example; `variant="Detail.*"` would restrict available templates to any pagelet template whose name starts with Detail.

21.2.2.2 Adjusting the Slot Title

Rather than showing the value of `slotname` in the blue overlay, which is typically a technical string meaningful only to developers, you can replace the value with any string.

To do this:

- Add a `title` attribute to the `insite:calltemplate` tag as shown in this code:

```
<insite:calltemplate
  slotname="HelloSlot"
  tname="HelloDetail"
  args="c,cid"
  variant="HelloDetail|Detail"
  title="Article Detail Area" />
```

  Defining the `title` attribute overrides the default slot title.

21.2.2.3 Controlling Template Arguments

This section provides an example of controlling template arguments. We modify the HelloDetail template to accept an extra argument called `image-align` which will be used to align the article image left or right.

The process is as follows:

- The `image-align` argument has to be registered as a legal argument for the HelloDetail template. If this step is skipped, contributors are not able to set its value from the editorial UI.

- To ensure caching works properly, the new argument has to be declared as a cache criteria.

- Finally, the template code is modified to use the newly defined argument.

> **Note:**
>
> **On Using Eclipse with the WebCenter Sites Developer Tools plug-in**: When editing a Template asset from the Admin interface, if this Template is also opened in WSDT at the same time, you must remember to synchronize your changes to the WSDT workspace. Template metadata stored in the WSDT workspace will otherwise override the values entered from the web interface.

1. Declare `image-align` as a legal argument of our HelloDetail template asset.

    a. From the Admin interface, edit the HelloDetail template asset.

    b. For **Legal Arguments**, enter `image-align` and click **Add Argument**.

    c. Select **Required**.

    d. For **Argument Description** enter: `Image Alignment`.

    e. For **LegalValues**, add the following descriptions:

        • For value left, enter `Value Description: Aligned Left`.

        • For value right, enter `Value Description: Aligned Right`.

    f. Click **Save**.

2. Remember to sync the change made in WebCenter Sites with the WSDT workspace.

3. Use the WebCenter Sites Developer Tools plug-in to add `image-align` to the set of cache criteria.

    a. Right-click the HelloDetail Template in the WebCenter Sites workspace.

    b. Select **Properties**.

    c. In the **Cache Criteria** field, append `image-align` to the end of the list.

    d. Click **Submit**.

4. Optionally, a default value could be defined by using the **Additional element parameters** field and specifying the following value, for instance: `image-align=right`.

5. Modify the HelloDetail pagelet template code:

```
<insite:edit
  field="largeThumbnail"
  assettype="AVIImage"
  assetid='<%=ics.GetVar("imageId")%>' >

    <img class='photo <ics:getvar name="image-align"/>'
      src='<ics:getvar name="imageURL" />' />
</insite:edit>
...
```

Note that, in this particular case, the value of the parameter is used to set a different CSS class.

When you go to the slot properties panel, assuming HelloDetail is the currently selected layout, the **Advanced** tab now shows the alignment options.

21.2.3 Editing Presentation and Content Simultaneously

The `insite:calltemplate` tag allows editorial users to edit associated content and edit the layout. This section explains the difference between a content-editable slot and a presentation-editable slot and how to combine the functionality of both to allow editorial users to edit both the associated content and the template used to render the content.

This section includes the following topics:

- Understanding Content-Editable Slots and Presentation-Editable Slots

- Combining Content-Editable Slots and Presentation-Editable Slots

21.2.3.1 Understanding Content-Editable Slots and Presentation-Editable Slots

Content-editable slots allow users to edit associated content by providing a droppable zone for the user. Presentation-editable slots allow users to select a different template to render the content.

To create a content-editable slot (creates a droppable zone for the user) the `insite:calltemplate` tag is used with the following defined parameters:

- `assetid`: The edited asset ID.

- `assettype`: The edited asset type.

- `field`: The edited field.

- `cid`: The ID of the asset to be rendered by the called template.

- `c`: The asset type to be rendered by the called template.

- `tname`: The pagelet template used to render the associated asset.

This code defines a content-editable slot that creates a droppable zone for the user:

```
<insite:calltemplate
  assetid=" "
  assettype=" "
  field=" "
  cid=" "
  c=" "
  tname=" "
/>
```

To create a presentation-editable slot (allows users to select a different template to render content) the `insite:calltemplate` tag is used with the following defined parameters:

- `slotname`: This attribute defines an identifier for the slot that is being filled with the called template. It should be reasonably easy to understand and should be unique across all templates.

- `cid`: The id of the asset to be rendered by the called template.

- `c`: The asset type to be rendered by the called template.

• `tname`: The default pagelet template to be called.

This code defines a presentation-editable slot that allows users to select a different template to render the content:

```
<insite:calltemplate
  slotname=" "
  cid=" "
  c=" "
  tname=" "
/>
```

21.2.3.2 Combining Content-Editable Slots and Presentation-Editable Slots

You can combine the functionality of a content-editable slot and a presentation-editable slot to allow editorial users to edit both the associated content and the template used to render the content.

To combine the functionality of both content-editable slots and presentation-editable slots, the `insite:calltemplate` tag is used along with all the attributes required for both a content-editable slot and a presentation-editable slot.

• These attributes are required for a content-editable slot (creates a droppable zone for the user):

  `field, assetid, assettype`

• This attribute is required to define a presentation-editable slot (allows users to select a different template to render the content):

  `slotname`

This code combines the attributes for a content-editable slot and a presentation-editable slot:

```
<insite:calltemplate
  slotname=" "
  assetid=" "
  assettype=" "
  field=" "
  cid=" "
  c=" "
  tname=" "
/>
```

Combining a Content-Editable Slot and a Presentation-Editable Slot

This section builds on the `HelloSideBar` template created in Coding Templates for In-Context Content Editing.

Previously, the `HelloSideBar` template was coded as shown:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"%>
<%@ taglib prefix="assetset" uri="futuretense_cs/assetset.tld"%>
<%@ taglib prefix="ics" uri="futuretense_cs/ics.tld"%>
<%@ taglib prefix="render" uri="futuretense_cs/render.tld"%>
<%@ taglib prefix="insite" uri="futuretense_cs/insite.tld"%>

<cs:ftcs>

<render:logdep cid='<%=ics.GetVar("tid")%>' c="Template"/>
<assetset:setasset name="article"
  type='<%=ics.GetVar("c") %>' id='<%=ics.GetVar("cid") %>' />
```

```
<assetset:getattributevalues name="article"
  listvarname="relatedStories" attribute="relatedStories"
  typename="ContentAttribute" />

<insite:slotlist field="relatedStories">
  <ics:listloop listname="relatedStories">
    <ics:listget listname="relatedStories" fieldname="value"
      output="articleId" />

    <insite:calltemplate
      tname="Summary/SideBar"
      c="Article"
      cid='<%=ics.GetVar("articleId") %>' />

  </ics:listloop>
</insite:slotlist>
</cs:ftcs>
...
```

In this template, the related articles are made editable with content-editable slots (drop zones). That is, they are rendered using the Summary/SideBar template, without any possibility for editorial users to select a different template. However, they cannot change how the related article should be rendered. For this to happen, the `HelloSideBar` template needs to be modified as follows:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"%>
<%@ taglib prefix="assetset" uri="futuretense_cs/assetset.tld"%>
<%@ taglib prefix="ics" uri="futuretense_cs/ics.tld"%>
<%@ taglib prefix="render" uri="futuretense_cs/render.tld"%>
<%@ taglib prefix="insite" uri="futuretense_cs/insite.tld"%>

<cs:ftcs>

<render:logdep cid='<%=ics.GetVar("tid")%>' c="Template"/>
<assetset:setasset name="article"
  type='<%=ics.GetVar("c") %>' id='<%=ics.GetVar("cid") %>' />
<assetset:getattributevalues name="article"
  listvarname="relatedStories" attribute="relatedStories"
  typename="ContentAttribute" />

  <insite:slotlist
    slotname="RelatedStoriesSlot"
    field="relatedStories">

    <ics:listloop listname="relatedStories">
      <ics:listget listname="relatedStories" fieldname="value"
        output="articleId" />

      <insite:calltemplate
        tname="Summary/SideBar"
        c="Article"
        cid='<%=ics.GetVar("articleId") %>'
        variant="Summary.*" />

    </ics:listloop>
  </insite:slotlist>
</cs:ftcs>
```

This is the same code used previously, except that we have specified a `slotname` attribute, and a `variant` attribute (in this case, the list of available templates is restricted to all pagelet templates starting with Summary).

Because `slotname` was included inside the `insite:slotlist` tag, it is not required to add it for the inner `insite:calltemplate` tag. The value is automatically inherited, as is the value of `tname` and `field`.

For example:

```
<insite:slotlist slotname=" " field=" ">
  <insite:calltemplate tname=" " c=" " cid=" " />
</insite:slotlist>
```

The Summary/SideBar now behaves as a default template for the related articles. By right-clicking any related article and selecting the Change Content Layout feature, it now becomes possible to select alternate templates.

21.2.4 Understanding the Context System Variable

In WebCenter Sites templates, `context` is a system variable maintained internally. Its value is retrieved by using the `ics:getvar` JSP tag or the `ics.GetVar()` method.

For example:

```
<ics:getvar name=context/>
ics.GetVar(context)
```

The value of context is determined by default. It is initially set to an empty string. Then, for every template called using `render:calltemplate` or `insite:calltemplate`, the value of `context` changes in the called template following this logic:

```
if parent_context is empty
  context = <c>:<cid>:<tname>
otherwise
  context = <parent_context>;<c>:<cid>:<tname>
```

This section includes the following topics:

- About Defining the Scope of the Slot

- Using the Context Variable in Action

- Initializing the Context Value

- Overriding Context

- Caching Context

21.2.4.1 About Defining the Scope of the Slot

When defining a slot, it is possible for template developers to decide the scope of the slot. Typically, whether any presentation change made in this slot should be local or global.

- **local**: visible only on the currently edited web page

- **global**: spanning across multiple web pages in a site (possibly, ALL web pages in a site)

This is done by manipulating the value of the `context` variable and will be explained in the following sections.

21.2.4.2 Using the Context Variable in Action

WebCenter Sites stores presentation changes by recording the newly selected template against these items:

- Slot name

- Current site name

- Context

To modify the slot content layout:

1. Assign the `HelloArticleLayout` template to two avisports articles (for example,

    All 25 Nevada resorts serving great snow and Cold snap back on the scene).

2. Observe both of these articles in Web Mode of the Contributor interface.

    The main slot is rendered with the default template `HelloDetail` as this is the template specified as the default template in the JSP code.

3. Assign the Detail template to all 25 Nevada resorts serving great snow article using the Change Content Layout option.

4. Refresh the web page with the Cold snap back on the scene article. The presentation of the main slot has also been modified on this web page as it is also using the Detail template.

Consequently, when we modified the slot content layout from `HelloDetail` to `Detail`, the following presentation data was recorded:

- **site**: avisports

- **slotname**: ArticleDetail

- **context**: (empty)

- **tname**: Detail

Context is empty since, when the `HelloArticleLayout` template is executed, context is initially empty, and is never modified when the slot gets rendered. Thus, any web page of avisports rendered using the `HelloArticleLayout` template will match the recorded presentation data above. Consequently, the `Detail` template is now used for all article pages.

21.2.4.3 Initializing the Context Value

The behavior observed in the previous section may be the intended behavior, but in some cases, editorial users will have to be able to make local presentation changes, that is, changes visible only on the current web page being edited.

To do this, the `context` variable has to be set to a value which will uniquely identify a given web page. In our case, it is enough to initialize context with, for example, the template name, and the identifier and type of the rendered asset:

```
<ics:setvar
  name="context"
  value='<%=ics.GetVar("c")
  + ":" + ics.GetVar("cid")
  + ":HelloArticleLayout"%>' />
```

Other parameters can be added to initialize the context, depending on the intended result. We can add the line above to the `HelloArticleLayout` template and verify that presentation changes are *local* to each article page.

21.2.4.4 Overriding Context

Both the `render:calltemplate` tag and the `insite:calltemplate` tag have an optional `context` attribute, which can be used to override the current context.

21.2.4.5 Caching Context

Context is useful only when presentation editing capabilities are enabled on your site.

If that is not the case, context can be removed from every template's cache criteria (avoiding the creation of unnecessary duplicates in the page cache).

21.2.5 Using Slots with CSElement and SiteEntry Assets

In our previous examples, slots were used to hold content. In this section, we look at using slots to hold functionalities such as a navigation bar, a login box, a code snippet showing the last ten published articles in a site, and so on. These types of functionalities can be made available as a CSElement or SiteEntry asset.

By allowing CSElement or SiteEntry assets to be dropped in slots, non-technical users are given the ability to modify the behavior of a particular web page without having to modify code.

This section includes the following topics:

- Defining a Slot Containing a CSElement Asset

- When to Use CSElement or SiteEntry Assets

- About Defining Legal Arguments

- Consideration About Using Nested Slots

21.2.5.1 Defining a Slot Containing a CSElement Asset

A slot meant to contain a CSElement asset is typically defined as:

```
<insite:calltemplate
  slotname="Navbar"
  clegal="CSElement"
/>
```

Or can be defined in this manner if the slot is meant to show the same content across all pages of the site:

```
<insite:calltemplate
  slotname="Navbar"
  clegal="CSElement"
  context="Global"
/>
```

There is no need to specify a `tname` attribute in this case, since the CSElement and SiteEntry assets are directly referring to the JSP element in charge of rendering them.

To see SiteEntry or CSElement in the list of allowed asset types for the slot, you have to ensure both asset types have their **Can Be Child Asset** flag set to **True** (which means that this asset type can be the child asset type in an association field for another asset type.). See Configuring the Asset Type in Creating Basic Asset Types.

> **Note:**
>
> To see SiteEntry or CSElement in the list of allowed asset types for the slot, you have to ensure both asset types have their **Can Be Child Asset** flag set to **True** (which means that this asset type can be the child asset type in an association field for another asset type.). See Designing Basic Asset Types.

21.2.5.2 When to Use CSElement or SiteEntry Assets

A SiteEntry asset does not hold any code, but simply points at a CSElement asset. The only difference between one case and the other is that, when using a SiteEntry, the element is invoked through the cache engine. Consequently, the same result can be achieved by dropping a SiteEntry asset or dropping its related CSElement asset.

The decision to use SiteEntry or CSElement is therefore implementation-dependent. Exposing a functionality as a SiteEntry only will ensure that caching is used to render this particular code snippet.

21.2.5.3 About Defining Legal Arguments

Like templates, it is possible to define legal arguments for CSElement assets. Once a CSElement has been dropped in a slot, the legal arguments are accessible from the slot properties panel.

The same applies to dropping a SiteEntry asset except that the legal arguments shown in the slot properties panel are the legal arguments of the related CSElement asset. (SiteEntry assets do not have their own legal arguments.)

21.2.5.4 Consideration About Using Nested Slots

When you drop an asset in a content-editable slot, the template rendering this asset can potentially define other slots. To avoid too many controls and slots being rendered in a given area, the behavior of nested slots is automatically degraded to a simple droppable area (without a toolbar and overlay). It is generally recommended to avoid nested slots, to keep the Contributor interface simple and usable for users.

21.2.6 Constraining Asset Types

By default, WebCenter Sites allows the following asset types to be dropped into a slot:

- if `field` is defined (content-editable slot): any legal asset types given by the field definition.

- if `field` is not defined (presentation-editable slot): any asset type for which the **Can Be Child Asset** flag is set to **True**.

It is also possible to further restrict allowable asset types, using the `clegal` attribute:

```
<insite:calltemplate
  slotname="Main"
  clegal="Article,Product"
  ...
/>
```

The following syntax allows to additionally restrict by asset subtypes:

```
<insite:calltemplate
  slotname="Main"
  clegal="type1:subtype1,type2:subtype2"
```

```
   ...
/>
```

> **Note:**
>
> Using `"type:*"` (with asterisk as wildcard) is also valid, and behaves as the `"type"` value.

21.2.7 Preventing CSS and JavaScript Conflicts

The in-context UI is injecting styles and JavaScript into web pages, when rendered in the Contributor interface in **Web Mode / Edit View** (and only in this case).

This may possibly result in:

- **CSS conflicts**: for instance, slots are improperly displayed due to a site CSS rule applying to them)

- **JavaScript conflicts**: the web page has JavaScript which conflicts with the JavaScript injected in the page in the editing view. As a result, either the editorial UI or the page itself do not work properly.

CSS conflicts are typically solved by adding extra CSS rules, which are loaded only when the template is rendered inside the editorial UI, in editing mode. This is done by using the `insite:ifedit` tag, which allows to execute JSP code only when the JSP template is ran in editing mode:

```
<insite:ifedit>
<%-- This stylesheet import will only occur in editing mode.
  -- It will not have any impact on the actual rendering of the
  -- live site.
  --%>
  <link rel="stylesheet"
   type="text/css"
   href="/css/editorial.css" />
</insite:ifedit>
```

Extra CSS classes can be added to slots by using the `cssstyle` attribute of the `insite:calltemplate` tag, to specify specific CSS rules for slots.

CSS conflicts typically when rendering slots around elements which are floated: in this case, the blue or green overlay which is marking the slot area is likely to not have the proper dimension.

JavaScript conflicts are normally solved by degrading the behavior of the web page in editing mode only, to let the scripts injected by the editorial UI function properly. This may mean disabling some page functionality in **Web Mode / Edit View** of the Contributor interface.

21.3 Enabling Content Creation for Web Mode

All you need to do is provide a **Start Menu** item of type `insite`, at least one layout template applicable to the asset type the contributors are using, editing-specific presentation logic. You can make more changes according to what the contributors need.

See these topics:

- Defining a Start Menu for In-Context Creation

21.3.1 Defining a Start Menu for In-Context Creation

To enable a particular asset type for in-context creation, a start menu of type `insite` has to be provided. That is, the **New Insite** option has to be selected in the **Type** drop-down menu.

In addition, if the asset being created has one or more required fields, default values have to be provided in the start menu, using the **Default Values** menu.

If required values are missing, the user will be directed to the asset form.

When a user selects a **New Insite** start menu, the editorial user is first prompted to select a layout template and a name for the newly created asset.

Once you click **Continue**, if no workflow is configured, a new asset is created in the background and then rendered using the selected layout template.

21.3.2 Providing Layout Templates for In-Context Creation

The same layout templates used for editing can be used for in-context creation. However, extra care must be taken to make sure the template will render properly with an empty asset.

This typically requires the following:

- Style adjustments, so the various elements on the page are rendered in the appropriate positions, although no values are rendered

- Meaningful help text (no value indicators), displayed to users when an editable field is empty

- Additional presentation logic, only executed when the template is ran in editing mode, on the editorial platform

This section includes the following topics:

- [Adjusting Stylesheets](#)
- [Adjusting Stylesheets for Slots](#)

21.3.2.1 Adjusting Stylesheets

When the stylesheets have to be specifically adjusted for creating or editing content, the corresponding import statements can be enclosed in an `insite:ifedit` tag:

```
// import "delivery" stylesheets
<link type="text/css" rel="stylesheet" href="somecss.css" />
...

// then import "editing only" stylesheets. using the insite:ifedit
// tag ensures that only those stylesheets will be imported
// when rendering the template in create/edit mode.

<insite:ifedit>
  <link type="text/css" rel="stylesheet" href="edit.css"  />
```

```
   ...
</insite:ifedit>
```

21.3.2.2 Adjusting Stylesheets for Slots

When you adjust styles for the rendering of slots, the `cssstyle` attribute of the `insite:calltemplate` tag can be used to specify additional class names, which can then be used in CSS rules.

```
<insite:calltemplate
  slotname="mySlot"
  ...
  cssstyle="myClassName"
  ...
/>
```

For example, if we wanted to force `mySlot` to have a 50px height when it is empty, we could provide the following CSS rule:

```
.myClassName .emptyIndicator {height: 50px !important;}
```

CSS rules can be grouped inside a specific stylesheet, and then imported conditionally, only when the template is executed in the context of **Web Mode / Edit View** of the Contributor interface, using the `insite:ifedit` JSP tag.

21.3.3 Providing Empty Value Indicators

For slots, use the `emptytext` attribute of the `insite:calltemplate` tag:

```
<insite:calltemplate
  slotname="mySlot"
  emptytext="Drag an Article here"
  ...
/>
```

For other editing fields, use the `noValueIndicator` parameter of the `insite:edit` tag:

```
<insite:edit
  field="headline
  params="{noValueIndicator: 'Enter Headline Here'}"
  ...
/>
```

21.3.4 Providing Editing-Specific Presentation Logic

In some cases, the presentation logic will be slightly different in delivery mode, compared to create/edit mode.

For example, in create/edit mode, when rendering an in-context editable list of articles, you would want to always display 5 extra empty slots. In that case, the logic has to account for both delivery and edit mode. It could be written as follows:

```
<%
// assuming that "relatedArticles" contains a list of
// related articles
%>
<insite:slotlist field="someAssetField">
  <ics:listloop listname="relatedArticles">
    <ics:listget
      listname="relatedArticles"
```

```
        fieldname="value"
        output="articleid" />
      <div class="post">
        <insite:calltemplate
          tname="Summary"
          c="Article"
          cid='<%=ics.GetVar("articleid")%>' />
      </div>
    </ics:listloop>
</insite:slotlist>

<%
// in this example, we add five extra empty slots
%>
<insite:ifedit>
  <%
    // To not disrupt rendering in delivery, this code is
    // added inside the insite:ifedit tag
  %>
  <c:forEach begin="0" end="4">
    <div class="post">
      <%
      // no c, cid specified, this renders an empty slot
      %>
      <insite:calltemplate tname="Summary" />
    </div>
  </c:forEach>
</insite:ifedit>
```

Obviously, the code could be adjusted to accommodate for any particular logic. For instance, to display a maximum of 5 slots, empty or not. In that case, the last part of the code snippet could be written as follows:

```
<%
// get how many articles are in the list
%>
<ics:listget
  listname="relatedArticles"
  fieldname="#numRows"
  output="nbArticles" />

<c:forEach
  begin='<%=Integer.valueOf(ics.GetVar("nbArticles"))%>'
  end='4'>
  <div class="post">
    <insite:calltemplate tname='Summary' />
  </div>
</c:forEach>
```

22

Template Element Examples for Basic Assets

You may use some examples to develop a deeper understanding of coding templates for basic assets. These assets are collection, query, and article. With the help of these examples, you can also create site navigations for basic assets.

Topics:

- Creating Basic Modular Design

- Coding Links to the Article Assets in a Collection Asset

- Using the ct Variable

- Coding Templates for Query Assets

- Displaying an Article Asset Without a Template

- Displaying Site Navigation Information

- Displaying Non-Asset Information

The examples in these topics illustrate the information presented in Coding Elements for Templates and CSElements

22.1 Creating Basic Modular Design

How would you create an area of a Home page without writing a new code? A modular site design takes advantage of common elements by reusing them in several locations or contexts.

Create an area of a Home page from five separate elements. The following column on a site's home page displays the main stories of the day. There is a summary paragraph and byline for each story in the list. The titles of the stories are hyperlinks to the full story. Several of the stories, including the first story in the list, also present a photo:

Figure 22-1   Main Stories of the Day

This example describes how the first story in the list is identified, selected, positioned at the top of the list, and formatted.

These are the elements used to format the first story in the list:

- `FiscalNews/Page/Home`

- `FiscalNews/Collection/MainStoryList`

- `FiscalNews/Article/LeadSummary`

- `FiscalNews/ImageFile/TeaserSummary`

Topics:

- Home Element

- MainStoryList Element

- LeadSummary Element

- TeaserSummary Element

- Back to LeadSummary

- Back to MainStoryList

- Back to Home

22.1.1 Home Element

For the home page of a site, you can use a `Template` that is named `Home`. You can examine your `Template` in two ways:

- Search for and then inspect it in the Admin interface.

- Use Oracle WebCenter Sites Explorer to open the `Template` element.

For example, the `Template` for the home page of the Fiscal News site is here:

```
ElementCatalog/FiscalNews/Page/Home
```

When opened, the `Home` element first loads the `Home` page asset, names it `HomePage`, and then scatters the asset information into its fields:

```
<ASSET.LOAD TYPE="Variables.c" OBJECTID="Variables.cid" NAME="HomePage"/>
<ASSET.SCATTER NAME="HomePage" PREFIX="asset"/>
```

The value for `cid` is passed in from the Fiscal News URL, and the value for `c` is available because it is set as a variable in the `resarg1` column in the `SiteCatalog` page entry for the `Home Template`. This `Template` includes the following `ASSET.CHILDREN` tag:

```
<ASSET.CHILDREN NAME=HomePage LIST=MainStories CODE=TopStories/>
```

With this code, `Home` obtains a collection asset identified as the page asset's `TopStories` collection (`CODE=TopStories`) and creates a list named `MainStories` to hold it (`LIST=MainStories`).

Next, `Home` determines whether it successfully obtained the collection and then calls for the page entry of the `MainStoryList Template`.

```
<IF COND = IsList.MainStories=true>
<THEN>
<RENDER.SATELLITEPAGE pagename=FiscalNews/Collection/MainStoryList
ARGS_cid=MainStories.oid
ARGS_p=Variables.asset:id/>
<THEN/>
<IF/>
```

Notice that `Home` passes the identity of the list that holds the collection to `MainStories` with `ARGS_cid` and the identity of the `Home` page asset with `ARGS_p=Variables.asset:id`.

22.1.2 MainStoryList Element

The `MainStoryList` page entry invokes its root element:

```
ElementCatalog/FiscalNews/Collection/MainStoryList.xml
```

The `MainStoryList` element is the Template (root) element for the MainStoryList Template asset, which formats collection assets. This element creates the framework for the home page column that holds the main list of stories and then fills that column with the articles from the `TopStories` collection. `MainStoryList` uses two templates to format those articles:

- `LeadSummary` for the first article in the collection (the top-ranked article)

- `Summary` for the rest of the articles

This example discusses the `LeadSummary Template` element to describe how the first story in the list is displayed.

`MainStoryList` loads and scatters the collection that Home passed to it:

```
<ASSET.LOAD TYPE="Variables.c" OBJECTID="Variables.cid"
NAME="MainStoryListCollection"/>
<ASSET.SCATTER NAME="MainStoryListCollection" PREFIX="asset"/>
```

Then it extracts the articles from the collection and creates a list to hold them, ordering them by their rank:

```
<ASSET.CHILDREN NAME="MainStoryListCollection" LIST="theArticles"
ORDER="nrank" CODE=-/>
```

And then it calls for the page entry of the `LeadSummary Template`:

```
<RENDER.SATELLITEPAGE PAGENAME="FiscalNews/Article/LeadSummary"
ARGS_cid="theArticles.oid"
ARGS_ct="Full"
ARGS_p="Variables.p"/>
```

Once again, this element passes on several pieces of information:

- The identity of the list that holds the articles (`ARGS_cid`)

- The name of the template to use when creating the link to each of the articles (`ARGS_ct`)

- The identity of the originating page asset (`ARGS_p`), which is `Home`.

Because the list was ordered by rank and this code does not loop through the list, the value in `ARGS_cid` (`theArticles.oid`) is the object ID of the highest ranked article in the collection because that article is the first article in the list.

22.1.3 LeadSummary Element

The `LeadSummary` page entry invokes its root element (which is the `Template` element for the `LeadSummary Template`):

```
ElementCatalog/FiscalNews/Article/LeadSummary.xml
```

This element formats the first article in the `TopStories` collection, as follows:

- Retrieves the image file associated with the first article through the `TeaserImage` association.

- Invokes the `TeaserSummary` element to obtain the formatting code for the image.

- Uses a `RENDER.GETPAGEURL` tag to obtain the URL for the first article in the collection.

- Displays the `imagefile` asset, the title of the article as a hyperlink to the full article, the summary paragraph, and the byline.

First `LeadSummary` loads the article and names it `LeadSummaryArticle`:

```
<ASSET.LOAD TYPE="Variables.c" OBJECTID="Variables.cid" NAME="LeadSummaryArticle"/>
<ASSET.SCATTER NAME="LeadSummaryArticle" PREFIX="asset"/>
```

It obtains the assets associated with the article as its teaserimagefile asset, creating a list for that file named `TeaserImage`:

```
<ASSET.CHILDREN NAME="LeadSummaryArticle" LIST="TeaserImage" CODE="TeaserImageFile"/>
```

Finally, it calls the page entry for the `TeaserSummary Template`, passing it the ID of the `imagefile` asset held in the list:

```
<THEN>
<RENDER.SATELLITEPAGE PAGENAME="FiscalNews/ImageFile/TeaserSummary"
ARGS_cid="TeaserImage.oid"/>
```

```
</THEN>
</IF>
```

22.1.4 TeaserSummary Element

The `TeaserSummary` page entry invokes its root element, the `Template` element for the `TeaserSummary Template`:

```
ElementCatalog/FiscalNews/ImageFile/TeaserSummary
```

Because `imagefile` assets are blobs stored in the WebCenter Sites database, and blobs stored in the database must be served by the BlobServer servlet rather than the ContentServer servlet, this element obtains an HTML tag that uses a BlobServer URL.

For example, the `TeaserSummary Template` has the following `RENDER.SATELLITEBLOB` tag:

```
<RENDER.SATELLITEBLOB BLOBTABLE=ImageFile BLOBKEY=id BLOBCOL=urlpicture
BLOBWHERE=Variables.asset:id BLOBHEADER= Variables.asset:mimetype SERVICE=IMG SRC
ARGS_alt= Variables.asset:alttext ARGS_hspace=5 ARGS_vspace=5 />
```

The tag creates an HTML `<IMG SRC>` tag. The `SRC` is the blob in the `ImageFile` table identified through the ID passed in with `BLOBWHERE=Variables.asset:id`, and both its horizontal and vertical spacing are at five pixels.

When `TeaserSummary` is finished, `LeadSummary` continues.

22.1.5 Back to LeadSummary

When `LeadSummary` resumes, having obtained the teaser image for the first article in the `TopStories` collection, it uses `RENDER.GETPAGEURL` to obtain the URL for that article:

```
<RENDER.GETPAGEURL PAGENAME="FiscalNews/Article/Variables.ct"
cid="Variables.cid"
c="Article"
p="Variables.p"
OUTSTR="referURL"/>
```

Remember that when the `MainStoryList` element called the page entry for `LeadSummary`, it passed a `ct` variable set to `Full`. Therefore, the page name that `LeadSummary` is passing to `RENDER.GETPAGEURL` is really `FiscalNews/Article/Full`.

`RENDER.GETPAGEURL` creates the URL for the article based on the information passed in to it and then returns that URL to `LeadSummary` in a variable called `referURL`, as specified by the `OUTSTR` parameter.

`LeadSummary` uses the `referURL` variable in an HTML `<A HREF>` tag and then displays the link, the abstract of the article, and the byline:

```
<A class="featurehead" HREF="Variables.referURL" REPLACEALL="Variables.referURL">
<csvar NAME="Variables.asset:description"/></A>
 <BR/>
<span class="thumbtext"><csvar NAME="Variables.asset:abstract"/>
</span><BR/>
<span class="thumbcredit"><csvar NAME="Variables.asset:byline"/>
</span><BR/>
```

Note the use of the `REPLACEALL` tag as an attribute in the HTML `<A HREF>` tag. You must use this tag as an attribute when you want to use XML variables in HTML tags.

Now that `LeadSummary` is finished, `MainStoryList` continues.

22.1.6 Back to MainStoryList

Next `MainStoryList` loops through the rest of the articles in the `TopStories` collection and uses the `Summary Template` to format them.

```
ElementCatalog/Article/Summary
```

When `MainStoryList` is finished, `Home` continues.

22.1.7 Back to Home

Home resumes, with a call to the `WireFeedBox` page entry.

22.2 Coding Links to the Article Assets in a Collection Asset

When an element needs URLs to create a list of hyperlinks to dynamically served WebCenter Sites pages, you use the `RENDER.GETPAGEURL` tag to code links to the article assets in a collection asset, as this example shows.

This example refers to these elements:

- `ElementCatalog/FiscalNews/Page/SectionFront`

- `ElementCatalog/FiscalNews/Collection/PlainList`

For the purposes of this example, the code displayed is stripped of any error checking so that you can focus on how the links are created:

The following topics show how to code the links:

- SectionFront Element

- PlainList Element

22.2.1 SectionFront Element

`SectionFront` is the `Template` element, the root element, of the `SectionFront Template` that is assigned to the main section pages, such as News, Markets, and Stocks:

```
ElementCatalog/FiscalNews/Page/SectionFront.
```

One section of a page formatted with the `SectionFront` element displays a list of links to articles from the `SectionHighlights` collection that is associated with that page asset, as shown in the following figure:

Figure 22-2   List of Links to Articles

The SectionFront element is invoked when a visitor clicks a link to a section. First, SectionFront uses the variables c and cid to load and scatter the page asset, and names it SectionFrontPage:

```
<ASSET.LOAD TYPE="Variables.c" OBJECTID="Variables.cid"
NAME="SectionFrontPage"/>
<ASSET.SCATTER NAME="SectionFrontPage" PREFIX="asset"/>
```

The values for c and cid are passed to the SectionFront element from the link that invoked it. That link could be from the home page or any of several other locations.

After several ASSET.CHILDREN tags SectionFront has the following tag that retrieves the SectionHighlights collection:

```
<ASSET.CHILDREN NAME="SectionFrontPage" LIST="SectionHighlights"
CODE="SectionHighlight"/>
```

This code retrieves the collection with the CODE=SectionHighlights statement and stores it as a list, also named SectionHighlights.

Then SectionFront calls the page entry of the PlainList template (a collection template):

```
<RENDER.SATELLITEPAGE
pagename="FiscalNews/Collection/PlainList"
ARGS_cid="SectionHighlights.oid" ARGS_p="Variables.asset:id"/>
```

This code passes in the ID of the `SectionHighlights` collection (`cid`) and the ID of the current page asset (`p`), which is the page asset assigned the name of `SectionFrontPage`.

22.2.2 PlainList Element

The `PlainList` page entry invokes its root element, the template element for the `PlainList` template:

```
ElementCatalog/FiscalNews/Collection/PlainList.
```

`PlainList` extracts the articles from the collection and presents them in a list, by their rank, with the subheadline of the article. This element assumes that the assets in the collection are articles. To load and scatter the collection. `PlainList` uses the values in `c` and `cid` (passed in from the `SectionFront` element) to load and scatter the collection:

```
<ASSET.LOAD TYPE="Variables.c" OBJECTID="Variables.cid" NAME="PlainListCollection"/>
<ASSET.SCATTER NAME="PlainListCollection" PREFIX="asset"/>
```

`PlainList` then sets the variable `ct` to `Full` because a value for this variable was not passed in (`Full` is the name of an article template):

```
<IF COND="IsVariable.ct!=true">
<THEN>
<SETVAR NAME="ct" VALUE="Full"/>
</THEN>
</IF>
```

Next `PlainList` creates a list of all the child articles in the collection asset, listing them by their rank, and naming the list `theArticles`.

```
<ASSET.CHILDREN NAME="PlainListCollection" LIST="theArticles" OBJECTTYPE="Article"
ORDER="nrank" CODE=-/>
```

Note that this `ASSET.CHILDREN` tag used the `OBJECTTYPE` parameter. If you use the `OBJECTTYPE` parameter with this tag, the resulting list of children is a join of the `AssetRelationTree` and the asset table for the type you specified (in this case, the `Article` table), and it contains data from both tables.

There is now no need for subsequent `ASSET.LOAD` tags because the data that the `PlainList` element is going to use to create the links to these articles is stored in the `Article` table.

`PlainList` loops through the list of articles, using the `RENDER.GETPAGEURL` tag to create a URL for each one. In this case (because the code does not use subsequent `ASSET.LOAD` tags for each of the children assets) the element includes a `RENDER.LOGDEP` tag in the loop:

```
<LOOP LIST="theArticles">
<RENDER.LOGDEP cid="theArticles.id" c="Article"/>
<RENDER.GETPAGEURL PAGENAME="FiscalNews/Article/
Variables.ct"
cid="theArticles.id"
c="Article"
p="Variables.p"
OUTSTR="referURL"/>
```

`PlainList` passes a `cid`, `pagename`, and the asset type with `ctype` for each article in the collection to the `RENDER.GETPAGEURL` tag. Because the variable `ct` was set to

Full, the page name being passed to the tag is actually `FiscalNews/Article/Full`.

The `RENDER.GETPAGEURL` tag returns a `referURL` variable for each article in the collection, as specified by the `OUTSTR` parameter, and then `PlainList` uses the value in the `referURL` variable to create an HTML `<A HREF>` link for each article.

Because the `ASSET.CHILDREN` tag that obtained this collection created a join between `AssetRelationTree` and the `Article` table, `PlainList` can use the article's subheadline field to create the link:

```
<A class="wirelink" HREF="Variables.referURL"
 REPLACEALL="Variables.referURL">
<csvar NAME="Variables.theArticles:subheadline"/>
</A>
</LOOP>
```

Note the use of the `REPLACEALL` tag as an attribute for this HTML tag. You must use this tag as an HTML attribute when you want to use XML variables in an HTML tag. See the *Oracle Fusion Middleware Tag Reference for Oracle WebCenter Sites Reference*.

22.3 Using the ct Variable

Sometimes you want to use a template other than an asset's default template. In such a case, you supply the name of an alternate template with the `ct` variable.

Assets are assigned a template when they are created, the identity of an asset's template (which is not the same as a default approval template) is part of the information you obtain with an `ASSET.LOAD` or `ASSET.CHILDREN` tag. The `ct` variable is used in child templates or alternate templates. For example, when a visitor browses the Fiscal News site, there are text-only versions of most of the site available to that visitor. The text-only format is not the default format, and content providers do not assign text-only formats to their assets. The Fiscal News page elements are coded to provide the ID of the alternate, text-only template when it is appropriate to do so.

Every page on the site uses the same element, the `TextOnlyLink` element, to determine the URL embedded in the **Plain Text** link for that page. The `TextOnlyLink` element returns the correct URL for each page because the **Plain Text** link on each page passes the `TextOnly` element the information that it needs:

- The ID of the page making the request.

- The alternate, text-only template (that is, the child template) to use for the **Plain Text** link.

These elements are used in this example:

- `ElementCatalog/FiscalNews/Page/SectionFront`

- `ElementCatalog/FiscalNews/Page/SectionFrontText`

- `ElementCatalog/FiscalNews/Common/TextOnlyLink`

- `ElementCatalog/FiscalNews/Page/ColumnistFront`

See these topics that describe how to use the `ct` variable to specify alternate templates for displaying pages as plain text:

- SectionFront Element

- TextOnlyLink Element

- [ColumnistFront](#)

22.3.1 SectionFront Element

The `SectionFront` element for the Fiscal News site is here:

`ElementCatalog/FiscalNews/Page/SectionFront.`

`SectionFront` is the `Template` element (root element) of the `Template` asset assigned to the standard section pages on the site, pages such as News, Money, and Stocks.

This element includes a `CALLELEMENT` tag:

```
<CALLELEMENT NAME="FiscalNews/Common/TextOnlyLink">
<ARGUMENT NAME="ct" VALUE="SectionFrontText"/>
<ARGUMENT NAME="assettype" VALUE="Page"/>
</CALLELEMENT>
```

`TextOnlyLink` is the element that creates the **Plain Text** Link. `SectionFront` passes it the name of the alternate template (`ct=SectionFrontText`) and the name of the asset type (`assettype=Page`).

22.3.2 TextOnlyLink Element

The `TextOnlyLink` element is here:

`ElementCatalog/FiscalNews/Common/TextOnlyLink`

When `TextOnlyLink` executes, it checks to see whether there is a value for `ct`:

```
<IF COND="IsVariable.ct!=true">
<THEN>
<SETVAR NAME="ct" VALUE="Variables.asset:templateText"/>
</THEN>
</IF>
```

In this example, there is a value for `ct` because the `SectionFront` element passed in `ct=SectionFrontText`.

Next, `TextOnlyLink` uses a `RENDER.GETPAGEURL` tag to obtain a URL for the **Plain Text** link, passing in the page name by concatenating one based on the variables that were passed to `TextOnlyLink` by `SectionFront`.

```
<RENDER.GETPAGEURL PAGENAME="FiscalNews/Variables.assettype/Variables.ct"
cid="Variables.asset:id"
c="Variables.assettype"
p="Variables.p"
OUTSTR="referURL"/>
```

`TextOnlyLink` knows that `ct=SectionFrontText` and that `assettype=Page`. Therefore, `FiscalNews/Variables.assettype/Variables.ct` means `FiscalNews/Page/SectionFrontText`.

Now that `TextOnlyLink` has a URL (in the `referURL` variable specified by the `OUTSTR` parameter), it can create the **Plain Text** link with an HTML `<A HREF>` tag:

```
<A class="contentlink" HREF="Variables.referURL"
REPLACEALL="Variables.referURL">
<img src="/futuretense_cs/bf/images/TextOnly.gif" width="22"
height="14" border="0" HSPACE="3"/>Plain Text</A><BR/>
```

Note the use of the REPLACEALL tag as an attribute for this HTML tag. You must use this tag as an HTML attribute when you want to use XML variables in an HTML tag. See the *Oracle Fusion Middleware Tag Reference for Oracle WebCenter Sites Reference*.

And then TextOnlyLink clears the ct variable:

```
<REMOVEVAR NAME="ct"/>
```

When a visitor clicks the **Plain Text** link, the article is formatted with the SectionFrontText element and then displayed in the browser.

22.3.3 ColumnistFront

The ColumnistFront element is here:

```
ElementCatalog/FiscalNews/Page/ColumnistFront
```

This element formats the web format page that displays the stories supplied from the Fiscal News columnists.

To create the **Plain Text** link in the upper right corner of a section page, ColumnistFront calls TextOnlyLink:

```
<CALLELEMENT NAME="FiscalNews/Common/TextOnlyLink">
<ARGUMENT NAME="ct" VALUE="ColumnistFrontText"/>
<ARGUMENT NAME="assettype" VALUE="Page"/>
</CALLELEMENT>
```

Based on the information passed in from ColumnistFront, this time TextOnlyLink creates a **Plain Text** link that takes the visitor to FiscalNews/Page/ColumnistFrontText.

22.4 Coding Templates for Query Assets

To display assets of your choice, you use the standard WebCenter Sites element ExecuteQuery to run the Query asset.

Fiscal News uses several query assets. The following figure shows a query asset named Home Wire Feed, which is used to list wire-feed stories on the home page:

Figure 22-3 Query Asset Listing Wire Feed Stories
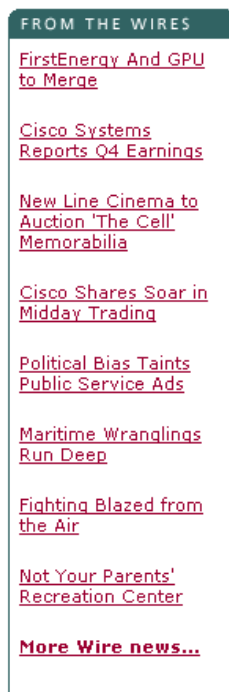
These elements are used in this example:

- `ElementCatalog/FiscalNews/Page/Home`

- `ElementCatalog/FiscalNews/Query/WireFeedBox`

- `ElementCatalog/OpenMarket/Xcelerate/AssetType/Query/ExecuteQuery`

See these topics on coding templates for Query assets:

- Home Element

- WireFeedBox Element

- ExecuteQuery Element

- Back to WireFeedBox

22.4.1 Home Element

The `Template` element for the `Home` page is here:

```
ElementCatalog/FiscalNews/Page/Home
```

When it runs, `Home` first loads the `Home` page asset, names it `HomePage`, and then scatters the asset information in its fields:

```
<ASSET.LOAD TYPE="Variables.c" OBJECTID="Variables.cid" NAME="HomePage"/>
<ASSET.SCATTER NAME="HomePage" PREFIX="asset"/>
```

The values for `c` and `cid` are passed in from the Fiscal News URL.

After several `CALLELEMENT` and `RENDER.SATELLITEPAGE` tags, `Home` includes the following `ASSET.CHILDREN` tag:

```
<ASSET.CHILDREN NAME="HomePage" LIST="WireFeedStories"
CODE="WireFeed"/>
```

Notice that in this line of code, the OBJECTTYPE parameter is not used. CODE=WireFeed is enough information for WebCenter Sites to locate and retrieve the query assigned to the HomePage asset through the WireFeed association, and there is no need to create a join between the AssetRelationTree and Query tables because all that Home needs is the ID of the query. The WireFeed query is retrieved and stored as WireFeedStories.

Next, Home calls the page entry of the WireFeedBox Template, passing it the cid of the query stored as WireFeedStories:

```
<RENDER.SATELLITEPAGE PAGENAME="FiscalNews/Query/WireFeedBox"
ARGS_cid="WireFeedStories.oid"
ARGS_p=Variables.asset:id/>
```

Home passes on several pieces of information: the identity of the query with the cid=WireFeedStories.oid statement and the identity of the originating page asset, Home, with the p=Variables.asset:id statement.

22.4.2 WireFeedBox Element

The WireFeedBox page entry invokes its root element, the template element for the WireFeedBox template:

```
ElementCatalog/FiscalNews/Query/WireFeedBox
```

This element invokes the ExecuteQuery element to run the query and then displays a list of links to the article assets returned by the query.

First, WireFeedBox loads the query asset passed in from Home, names it WireFeedBoxQuery, and then retrieves the values from all of its fields with an ASSET.SCATTER statement:

```
<ASSET.LOAD TYPE="Variables.c" OBJECTID="Variables.cid" NAME="WireFeedBoxQuery"/>
<ASSET.SCATTER NAME="WireFeedBox" PREFIX="asset"/>
```

Variables.cid is the WireFeedStories.oid passed in from the Home element.

Then WireFeedBox calls the ExecuteQuery element:

```
<CALLELEMENT NAME="OpenMarket/Xcelerate/AssetType/Query/ExecuteQuery">
<ARGUMENT NAME="list" VALUE="ArticlesFromWireQuery"/>
<ARGUMENT NAME="assetname" VALUE="WireFeedBoxQuery"/>
<ARGUMENT NAME="ResultLimit" VALUE="8"/>
</CALLELEMENT>
```

WireFeedBox passed in the query asset, the name of the list to create to hold the results of the query, and a limit of 8 so that no matter how many assets the query returns to the ExecuteQuery element, ExecuteQuery returns only 8 of them to WireFeedBox.

22.4.3 ExecuteQuery Element

The ExecuteQuery element runs the query asset:

```
ElementCatalog/OpenMarket/Xcelerate/AssetType/Query/ExecuteQuery
```

The query assets that can be assigned to a page asset as that page's Wire Feed query are coded to return field data rather than the IDs of assets only. Therefore,

ExecuteQuery returns up to 8 article assets and the data from several of their fields to WireFeedBox.

The first line of code in the element is RENDER.UNKNOWNDEPS because there is no way of knowing which assets will be returned, so there is no way to log dependencies for them.

When ExecuteQuery is finished, WireFeedBox resumes.

22.4.4 Back to WireFeedBox

WireFeedBox resumes, looping through the list of articles returned by ExecuteQuery, and obtaining a URL for each one by using a RENDER.GETPAGEURL tag.

Because there is no way of knowing which article assets will be returned by ExecuteQuery, there is a RENDER.FILTER tag included in the loop to filter out unapproved assets when the publishing method is Export to Disk:

```
<RENDER.FILTER LIST="ArticlesFromWireQuery" LISTVARNAME="ArticlesFromWireQuery"
LISTIDCOL="id"/>
<if COND="ArticlesFromWireQuery.#numRows!=0">
<then>
<LOOP LIST="ArticlesFromWireQuery">
<RENDER.GETPAGEURL PAGENAME="FiscalNews/Article/WireStory"
cid="ArticlesFromWireQuery.id"
c="Article"
p="Variables.p"
OUTSTR="referURL"/>
<A class="wirelink" HREF="Variables.referURL" REPLACEALL="Variables.referURL"><csvar
NAME="ArticlesFromWireQuery.subheadline"/></A><P/>
</LOOP>
</then>
</if>
```

The RENDER.GETPAGEURL tag returns a URL for each article in the list in a variable named referURL. WireFeedBox uses the value from the referURL variable to create links to the articles, using the content from their subheadline fields (which is one of the fields that the Wire Feed query returned) as the hyperlinked text.

Note the use of the REPLACEALL tag as an attribute for this HTML tag. You must use this tag as an HTML attribute when you want to use XML variables in an HTML tag. See the *Oracle Fusion Middleware Tag Reference for Oracle WebCenter Sites Reference*.

22.5 Displaying an Article Asset Without a Template

Full, AltVersionBlock, and EmailFront elements let you display an Article asset without a template.

This figure shows an example of an email form for an article, provided by Fiscal News with an *Email this article to a friend* function:

Figure 22-4   Email Form

Obviously the Fiscal News developers do not want the Fiscal News content providers to assign the email form to an article as the article's `Display Style` (`Template`). Therefore, there is no `Template` asset that points to the email element that creates the article email form.

These elements are used in this example:

- `ElementCatalog/FiscalNews/Article/Full`

- `ElementCatalog/FiscalNews/Article/AltVersionBlock`

- `ElementCatalog/FiscalNews/Util/EmailFront`

See these topics that describe how to display an article asset without a template:

- Full Element

- AltVersionBlock Element

- EmailFront Element

22.5.1 Full Element

The `Full Template` for articles is here:

`ElementCatalog/FiscalNews/Article/Full`

This element provides the formatting code for articles when they are displayed in full. It displays the following items:

- A site banner

- The left navigation column

- A collection of related stories

- The text of the article

- A photo for the article

- A link that prints the story

- A link that emails the story

After several `RENDER.SATELLITEPAGE` and `CALLELEMENT` tags, the `FULL` element includes the following tag:

```
<CALLELEMENT NAME="FiscalNews/Article/AltVersionBlock"/>
```

22.5.2 AltVersionBlock Element

For this example, the `AltVersionBlock` element, which can get the URL for the print version of an article or the email version, is here:

```
ElementCatalog/FiscalNews/Article/AltVersionBlock
```

`AltVersionBlock` is a short element with two `RENDER.GETPAGEURL` tags. The first `RENDER.GETPAGEURL` tag obtains the URL for the print version of an article. The second `RENDER.GETPAGEURL` tag obtains the URL for the email version of the story.

Because the Fiscal Newsl developers want a dynamic URL for the email version of the story even if the site is a static site, the second `RENDER.GETPAGEURL` tag uses the `DYNAMIC` parameter.

The second `RENDER.GETPAGEURL` tag has this code:

```
<RENDER.GETPAGEURL PAGENAME="FiscalNews/Util/EmailFront"
cid="Variables.asset:id"
c="Article"
DYNAMIC="true"
OUTSTR="referURL"/>
```

`AltVersionBlock` passes in the `pagename` for the `EmailFront` page entry, and a value for `c`, and `cid`, and sets the `DYNAMIC` parameter to `true`. The tag creates a dynamic URL for the article (even if the publishing method is Export to Disk) and returns it in a variable named `referURL`, as specified by the `OUTSTR` parameter.

22.5.3 EmailFront Element

In this example, `EmailFront` is the page name that `AltVersionBlock` passes to the `RENDER.GETPAGEURL` element. Because there is no corresponding template for `EmailFront`, WebCenter Sites would not create a page entry in the `SiteCatalog` for `EmailFront` by default. The Fiscal News developers created the `SiteCatalog` entry for this element manually through Oracle WebCenter Sites Explorer:

```
ElementCatalog/FiscalNews/Util/EmailFront
```

This element creates a form that displays the first paragraph of the article that the visitor has chosen to email.

First, `EmailFront` loads the article asset:

```
<ASSET.LOAD TYPE="Article" OBJECTID="Variables.cid" NAME="EmailFront"/>
<ASSET.SCATTER NAME="EmailFront" PREFIX="asset"/>
```

Then it formats several parts of the page before creating the email form, using the HTML `FORM` tag:

```
<FORM NAME="mailform" onSubmit="return checkEmail();" METHOD="POST" ACTION=...
```

EmailFront then calls the LeadSummary page entry to display a summary of the article in the form:

```
<RENDER.SATELLITEPAGE
ARGS_pagename="FiscalNews/Article/LeadSummary"
ARGS_cid="Variables.cid"
ARGS_ct="Full"
ARGS_p="Variables.p"/>
```

For information about the LeadSummary element, see Creating Basic Modular Design.

22.6 Displaying Site Navigation Information

To extract information from the SitePlanTree table, you use the WebCenter Sites SITEPLAN tag family. The navigation bar at the top of the Fiscal News home page is created by extracting the structure information from the SitePlanTree table.

These elements are used in this example:

- ElementCatalog/FiscalNews/Article/Home

- ElementCatalog/Pagelet/Common/SiteBanner

- ElementCatalog/FiscalNews/Site/TopSiteBar

See these topics that describe how to display site navigation information:

- Home Element

- SiteBanner Element

- TopSiteBar Element

- Back to SiteBanner

22.6.1 Home Element

You can use Oracle WebCenter Sites Explorer to open and examine the template element for the Home Template:

```
ElementCatalog/FiscalNews/Page/Home
```

The first RENDER.SATELLITEPAGE tag in this Template follows:

```
<RENDER.SATELLITEPAGE PAGENAME="FiscalNews/Pagelet/Common/SiteBanner"/>
```

22.6.2 SiteBanner Element

The SiteBanner pagelet, which invokes its root element for this example, is here:

```
ElementCatalog/FiscalNews/Common/SiteBanner
```

SiteBanner gathers the images for the banner (the Fiscal News logo and an advertising image) and then calls an element that creates the navigational links to the main sections of the site.

The SiteBanner element includes the CALLELEMENT tag:

```
<CALLELEMENT NAME="Fiscal News/Site/TopSiteBar"/>
```

22.6.3 TopSiteBar Element

The `TopSiteBar` element for this example is here:

```
ElementCatalog/FiscalNews/Site/TopSiteBar
```

`TopSiteBar` executes, creating the navigational links to the main sections in the Fiscal News site.

The following topics describe how to create links for pages on the site:

- Creating the Link for the Home Page
- Creating the Links to the Home Page's Child Pages

22.6.3.1 Creating the Link for the Home Page

To create the link for the Home page, first, `TopSiteBar` loads the Home page, names it `target`, gets the value from its ID field, and stores that value in the output variable `pageid`:

```
<ASSET.LOAD TYPE="Page" NAME="target" FIELD="name" VALUE="Home" DEPTYPE="exists"/>
<ASSET.GET NAME="target" FIELD="id" OUTPUT="pageid"/>
```

Note that the `ASSET.LOAD` tag changes the dependency type from its default of `exact` to `exists` with the `DEPTYPE` parameter. For a link like this one, a link in a navigational bar, it makes more sense for the dependency to be an `exists` dependency.

Then `TopSiteBar` uses the variable `pageid` to obtain a URL for the Home page from a `RENDER.GETPAGEURL` tag:

```
<RENDER.GETPAGEURL PAGENAME="FiscalNews/Page/Home"
cid="Variables.pageid"
c="Page"
OUTSTR="referURL"/
```

Next `TopSiteBar` extracts the page asset's name from its **Name** field and uses it as the text for the hyperlink:

```
<ASSET.GET NAME="target" FIELD="name" OUTPUT="thepagename"/>
<A class="sectionlinks" HREF="Variables.referURL"
REPLACEALL="Variables.referURL"><csvar NAME="Variables.thepagename"/></A>
```

Note the use of the `REPLACEALL` tag as an attribute for this HTML tag. You must use this tag as an HTML attribute when you want to use XML variables in an HTML tag. See Variables in HTML Tags.

22.6.3.2 Creating the Links to the Home Page's Child Pages

In the next part of the code, `TopSiteBar` creates links for the child pages of the `Home` page. To determine the child pages of the `Home` page, `TopSiteBar` must first determine the node ID of the `Home` page.

The node ID of a page asset is different from its object ID:

- You use an object ID to extract information about an asset from asset tables.
- You use a node ID to extract information about a page asset from the `SitePlanTree` table.

First, `TopSiteBar` determines the node ID of the `Home` page:

```
<ASSET.GETSITENODE NAME="target" OUTPUT="PageNodeId"/>
```

Then it uses that information to load the `Home` page as a `siteplan` node object:

```
<SITEPLAN.LOAD NAME="ParentNode" NODEID="Variables.PageNodeId"/>
```

With the `Home` page node identified and loaded, `TopSiteBar` can then obtain the `Home` page's child nodes, storing them in a list that it names `PeerPages`, and ordering them according to their rank:

```
<SITEPLAN.CHILDREN NAME="ParentNode" TYPE="PAGE" LIST="PeerPages" CODE="Placed"
ORDER="nrank"/>
```

And now `TopSiteBar` loops through all the child nodes at the first level, using the `RENDER.GETPAGEURL` tag to create a URL for the link to each page:

```
<IF COND="IsList.PeerPages=true">
<THEN>
<LOOP LIST="PeerPages"> | 
<ASSET.LOAD NAME="ThePage" TYPE="Page"
 OBJECTID="PeerPages.oid"/>
<ASSET.GET NAME="ThePage" FIELD="name"
OUTPUT="thepagename"/>
<ASSET.GET NAME="ThePage" FIELD="template"
OUTPUT="pagetemplate"/>
<RENDER.GETPAGEURL PAGENAME="FiscalNews/Page/
Variables.pagetemplate"
cid="PeerPages.oid"
c="Page"
OUTSTR="referURL"/>
<A class="sectionlinks" HREF="Variables.referURL"
REPLACEALL="Variables.referURL">
<csvar NAME="Variables.thepagename"/>
</A>
```

Notice how the page name is constructed in this example. The second `ASSET.GET` statement gets the name of the page's `Template` from its `template` field:

```
<ASSET.GET NAME="ThePage" FIELD="template"
OUTPUT="pagetemplate"/>
```

Then, that information is used in the `PAGENAME` parameter passed to the `RENDER.GETPAGEURL` tag:

```
PAGENAME="FiscalNews/Page/Variables.pagetemplate"/>
```

Therefore, if the template for the page asset is `SectionFront`, this argument statement passes `pagename=FiscalNews/Page/SectionFront`. And if the template for the page asset is `AboutUs`, this argument statement passes `pagename=FiscalNews/Page/AboutUs`.

22.6.4 Back to SiteBanner

SiteBanner is finished after the call to `TopSiteBar`. The `SiteBanner` element is invoked on each page in the site.

Because `SiteBanner` has a page entry in the `SiteCatalog` table, the results of the navigational bar that `TopSiteBar` creates is cached the first time a visitor requests a page on the Fiscal News site. This speeds up performance because the site does not

have to reinvoke the `TopSiteBar` element for each and every page that the visitor subsequently visits.

22.7 Displaying Non-Asset Information

WebCenter Sites includes elements that you can use for rendering and displaying information which is not stored as an asset in the WebCenter Sites database. For example, the Fiscal News site displays today's date on each page. The date is not an information that can be stored as an asset.

The elements are:

* `ElementCatalog/FiscalNews/Article/Home`

* `ElementCatalog/Common/ShowMainDate`

The following topics describe how to display non-asset information:

* Home Element

* ShowMainDate Element

22.7.1 Home Element

For this example, the `Template` element for the `Home Template` is here:

```
ElementCatalog/FiscalNews/Page/Home
```

The third `CALLELEMENT` tag in this element invokes the `ShowMainDate` element:

```
<CALLELEMENT NAME=FiscalNews/Common/ShowMainDate/>
```

22.7.2 ShowMainDate Element

The `ShowMainDate` element for this example is here:

```
ElementCatalog/FiscalNews/Common/ShowMainDate
```

`ShowMainDate` executes. The main line of code is this one:

```
 <span class="dateline"><csvar NAME="CS.Day CS.Mon CS.DDate, CS.Year"/></span>
```

It calculates the date and then returns that value to the `Home` element, which displays it at the top of the page, under the navigation bar and over the main list of stories.

This element performs a simple calculation and then outputs the value into the HTML code that is rendered in the browser window. There are no content assets that it formats or `Template` assets that use it as a root element. It also has no `SiteCatalog` entry because its result (the date) should be calculated each time the `Home` page is rendered.

23

Creating Collection Assets, Query Assets, and Page Assets

A collection asset stores an ordered list of assets of one type. A query asset stores a database query for retrieving a list of other assets from the database. Page assets are site design assets that store references to other assets in a designed fashion. Easy-to-use forms are available in the Admin interface for you to create collection, query, and page assets.

- About Creating Assets

- Creating Collection Assets

- Creating Query Assets

- Creating Page Assets

23.1 About Creating Assets

The core asset types delivered with WebCenter Sites provide the basic site design logic. Because you assign Template assets to your other assets, it is typical to create your templates before you create your site design asset types.

Coding Templates for In-Context and Presentation Editing describes how to create Template assets.

The procedures for working with assets of any type are very similar and are described thoroughly in Working with Assets in *Oracle Fusion Middleware Using Oracle WebCenter Sites*. This chapter presents procedures that are unique for the collection, query, and page asset types.

23.2 Creating Collection Assets

Content contributors use collection assets to store ordered lists of assets. Each collection asset can store a list of assets of one type only. You **create** (or design) a collection asset by naming it and selecting query assets for it. By default, you can select up to three query assets. You can create additional associations for additional queries if the need arises.

For information about creating associations, see Configuring Association Fields (Optional) in Creating Basic Asset Types.

A collection uses a query asset to obtain a list of possible assets for the collection. You **build** (or populate) a collection by running its queries, selecting assets from the results of the queries, and then ranking and ordering the assets that you selected. This ranked, ordered list is the collection.

Using collections is one way to keep the content displayed on rendered pages current and up-to-date. For example, you can have a site that uses several collections. If you

select a collection to be displayed on an asset, a publisher or content contributor can then change the content identified by that association by doing one of the following:

- Selecting a different collection from the tree

- Building the assigned collection and selecting different assets in it

This section includes the following topics:

- Before You Begin

- Creating a Collection Asset

- Sharing a Collection Asset

23.2.1 Before You Begin

Before you **create** collection assets, note the following:

- A collection must have at least one query, so be sure that you create the queries before you try to create your collections.

- Because you assign templates to collections, you should also create the Template assets before you create your collection assets.

Before you **build** the collection, you should determine how the Template asset assigned to it is coded. For example, if you select 100 assets for a collection but the template is coded to display only five of them, the following occurs:

- The rendered page that displays those assets displays only the first five.

- The page takes longer to render than necessary because WebCenter Sites has to sort through all 100 assets even though it displays only the first five.

For more information about building a collection, see Determining Function Privileges in *Oracle Fusion Middleware Administering Oracle WebCenter Sites*.

23.2.2 Creating a Collection Asset

You can create a collection asset through the Admin interface.

> **Note:**
>
> To use this procedure, you must have Collection asset types enabled for the site you are working in. Step 4 indicates whether they are enabled.

1. Open the Admin interface.

2. Ensure you have completed the steps in Before You Begin.

3. Click **New** on the button bar.

4. Select **New Collection** from the list of asset types. (If Collection asset types are not enabled, the option is not displayed.)

   The Collection form opens.

5. (Required) In the **Name** field, type a descriptive name for the page. You can enter up to 64 alphanumeric characters, but the first character must be a letter.

Underscores (_) and hyphens (-) are acceptable, but tab and space characters are not.

6. In the **Description** field, type a brief description of the collection. You can enter up to 128 characters.

7. (Required) In the **Subtype** field, select the type of asset this collection will hold.

> **Note:**
>
> Collection subtypes are controlled by Query. When a query is set up for a certain asset type, that asset type becomes a value of the **Subtype** field. The **SubType** field thus lists every asset type for which a query was created. See Creating a Query Asset.

8. In the **Select a Template** field, select a Template asset from the drop-down list.

9. In the **Category** field, select a category from the drop-down list. (If you do not select a category, the first item on the list is selected by default.)

10. In the **Keywords** field, enter keywords that you and others can use as search criteria in the Advanced Search form when you search for this collection in the future.

11. In the **Associated queries** section, select up to three queries. All of the queries that you select for this collection must return assets of the same type.

12. Click **Save**.

23.2.3 Sharing a Collection Asset

Before you share a collection asset, consider the following:

- Building a collection in one site builds it in all of the sites that it is shared with. You cannot build a collection to include different assets for different sites.

- The query assets used in the shared collection must be coded to return only assets that are shared to all the sites that the collection is shared with.

- As with any shared asset, be sure that the template assigned to the collection is also shared to the other site.

For the basic procedure on sharing assets, see Sharing Blog Assets Sharing Blog Assets in *Oracle Fusion Middleware Administering Oracle WebCenter Sites*.

23.3 Creating Query Assets

A query asset stores a database query that retrieves a list of other assets from the database. If you plan to use a query for a collection, remember that it returns assets of one type only.

See these topics about query assets:

- How to Use Query Assets and Other Assets

- How to Store the Query

- Commonly Used Fields for Queries

- Before You Begin Creating Query Assets

- Creating a Query Asset

- Sharing Query Assets

- Previewing and Approving Query Assets

23.3.1 How to Use Query Assets and Other Assets

WebCenter Sites uses queries differently in collection assets than it does for other assets:

- When you build (or populate) a collection, you run one or more query assets and then select and order the assets that you want from the resulting list. The collection is a **static** list of assets selected from the query resultsets.

- You can select queries for a page asset either through informal relationships or through named associations. You can select queries for other asset types (article, for example) through named associations.

  When the asset is rendered, it does not invoke the query directly. Either the template element that formats the asset or a template element that formats the query is coded to invoke a standard WebCenter Sites element called:

  ```
  OpenMarket/Xcelerate/AssetType/Query/ExecuteQuery
  ```

This element runs the query asset when the asset it is associated with is rendered, which means the resultset is **dynamic**.

23.3.2 How to Store the Query

A query asset can store its database query in one of two ways:

- **Directly**. You can write the query directly into the **SQL query** field of the Query form. You can either use standard SQL for the query, or, if your WebCenter Sites systems use an external search engine, you can use an appropriate search engine query.

- **Indirectly**. You can write the query in an element and then store the location of that element in the query asset by identifying it in the **Element name** field in the Query form. An element for a query is like any other element: you can use XML, JSP, JavaScript, HTML, and so on.

The following code is an example of a query named News Wire Feed which is stored directly; that is, the SQL query is written directly into the **SQL query** field in the Query form:

```
SELECT DISTINCT Article.id, Article.name, Article.updateddate,
Article.subheadline, Article.abstract, Article.description,
Category.description AS category, StatusCode.description AS
statusdesc FROM Article, Category, AssetPublication, StatusCode
WHERE Article.status!='VO' AND Article.category=Category.category
AND Article.status=StatusCode.statuscode AND Category.assettype='Article'
AND Article.source='WireFeed' AND Article.category='n' AND Article.id =
AssetPublication.assetid AND AssetPublication.pubid = 968251170475
ORDER BY Article.updateddate DESC
```

23.3.3 Commonly Used Fields for Queries

There are several WebCenter Sites fields, four of which are used in the preceding News Wire Feed query example, that you are likely to use in your queries:

- status

- updateddate

- category

- startdate and enddate

The rest of this section defines the fields in this list.

status

All assets have a `status`. When an asset is created, WebCenter Sites adds a row to the table that holds assets of that type and sets its status to PL, which means created.

This table lists and defines the status codes that WebCenter Sites uses.

Table 23-1   Status Codes

Status Code	Definition
PL	created
ED	edited
RF	received (from XMLPost, for example)
UP	upgraded from Xcelerate 2.2
VO	deleted (void)

These codes are listed in the `StatusCode` table in the database.

When an asset is deleted, WebCenter Sites changes its status to VO and renames the string in its **Name** field to its object ID.

Write your queries to exclude assets whose status is VO. For example: `WHERE Article.status!='VO'`

updateddate

The information in the **updateddate** field represents the date on which the information in the status field was changed to its current state. Depending on the design of your site, you can use a query to return assets based on this date.

category

The **category** is a default WebCenter Sites field that can categorize assets according to a convention that works for your sites. It is not required.

For example, if you had a banking site, you could have categories named Personal Finance, Banking and Loans, Rates and Bonds, News, and so on. You add categories for your sites on the Admin tab in the tree. See Configuring Categories (Optional) in Creating Basic Asset Types.

If you use category with your assets, you can write your queries to use category as a parameter. In the previously mentioned News Wire Feed query example, the `Article.category='n'` statement includes article assets from the News category.

pubid

A `pubid` is a unique value that identifies a site (or, in old terminology, a publication). When an asset is created, WebCenter Sites writes information about that asset to several database tables, one of which is the `AssetPublication` table.

An asset's row in the `AssetPublication` table includes the `pubid` of the site the asset was created for. If the asset is shared, the `AssetPublication` table has a row for each site that the asset is shared with. For example, if an article asset is available in two sites, there are two rows for that article in the `AssetPublication` table.

If you have only one WebCenter Sites site on your system or if your query results do not have to be site-specific, you do not have to code your queries to consider `pubid`. However, if you do not want your queries to return assets from another site, you can code your queries to restrict assets based on the `pubid` of the site.

startdate and enddate

Neither of the sample sites use the **startdate** and **enddate** fields but the WebCenter Sites database has columns to store this information. These fields exist so that you can assign time limits to assets. If your asset types use the startdate and enddate fields, you can create queries that select assets based on the dates stored in those fields.

23.3.4 Before You Begin Creating Query Assets

Before you begin creating query assets, consider the following:

- Query assets that are used on assets other than collections are not required to have templates. You can either create template elements specifically for your query assets that identify, run, and display the results, or you can code the template elements for your page assets to do that.

- When you write a query for a collection, be sure to code it to select the fields that are required for that asset type. WebCenter Sites is programmed to expect information from an asset type's required fields so that it can display that information in the Build Collection form.

- Query assets that are used only for collections have no need for templates. The template element assigned to the collection formats the assets in a collection's list of assets.

- For performance reasons, be sure to create efficient queries. For example:

  - Include as much logic as possible in the query rather than in the element that runs and displays the results of the query. For example, to filter or constrain a list of articles, be sure the query performs the filtering or constraining step so that the list returned to the element is complete rather than coding the query to return the entire list and using the element code to constrain the list.

  - Be sure your queries return only the information that the element displays.

- Query assets that are for collections must return assets of one type only.

23.3.5 Creating a Query Asset

You can create a query asset through the Admin interface.

1. Open the Admin interface.

2. Click **New** on the button bar.

3. Select **New Query** from the list of asset types. (Query asset types must be enabled for your site.)

   The Query form opens.

4. (Required) In the **Name** field, type a descriptive name for the query asset. You can enter up to 64 alphanumeric characters, but the first character must be a letter. Underscores (_) and hyphens (-) are acceptable, but tab and space characters are not.

5. In the **Description** field, type a brief description of the query. You can enter up to 128 characters.

6. In the **Template** field, select a Template asset from the drop-down list.

7. In the **Category** field, select a category from the drop-down list. (If you do not select a category, the first item on the list is selected by default.)

8. In the **Result of query** field, select the type of asset that this query returns. (The query can return assets of one type only if this asset is to be used by a collection.)

9. Do one of the following:

   - To store the query directly in this asset, select **Database**, and in the **SQL query** field write your query.

   - If you wrote the query in an element, select **Element** and then enter the entire name of the element in the **Element name** field.

10. Click **Save**.

23.3.6 Sharing Query Assets

If you plan to share a query asset with another site, consider the following tips:

- If you want your query results to be site-specific, be sure to include a `WHERE` clause for `pubid` so that the query does not return assets to a site where those assets have not been shared.

   - For example, in either a query for a collection or a query for a static site, you can use the following statement:

     `WHERE AssetPublication.pubid = SessionVariables.pubid`

     because `SessionVariables.pubid` is always set when you are building a collection or using the Export to Disk function.

   - If the query is to be used on a dynamic site, you can use that same statement if you code your elements to either pass in the identify of `pubid` to the `ExecuteQuery` element or to set the `SessionVariables.pubid` variable.

- Because page assets cannot be shared, you should not share query assets if they return page assets.

- As with any shared asset, if the query has a template, be sure that the template assigned to the query is also shared with the other site.

23.3.7 Previewing and Approving Query Assets

First, remember that not all query assets have their own templates. If a query asset was designed to be used on a page asset and it is the page asset's template that actually formats the query, you must preview the page to preview the query.

If your online site is a dynamic site (that is, you use the Mirror to Server publishing method) a query asset might return different assets on the management system than it does on the delivery system, depending on which assets have been published.

Therefore, if you preview your query to determine whether you should approve it or not, remember that the assets that it returns on the management system (where you are previewing it) could be different than the assets that it will return on the delivery system after it is published.

23.4 Creating Page Assets

Page assets are site design assets that you use to store references to other assets, organizing them according to the design that you and the other designers are implementing.

These page assets represent sections of the site, in essence the structure or organization of the site. They do not represent each and every rendered page that can possibly be served. This structural organization is primarily for the benefit of your WebCenter Sites users. This is not the only way of organizing your site, but it is convenient for your editors to see a structure that resembles your finished website.

Typically, you create page assets once: when you design the site. You associate collections, queries, articles, and so on with page assets and you code template elements that format the types of assets you want to associate with the page asset.

Before you can select the correct content for your page assets, you must be familiar with how your site is structured and what your template elements for page assets are designed to do. That is why you and other site developers (the people who are coding elements and creating Template assets) typically create the page assets for a site.

This section includes the following topics:

- Understanding the Page Asset Model

- How To Design Page Attributes

- How to Create a Page Asset

- How To Place Page Assets

- How To Move Page Assets in the Site Tree

- Considerations About Placing Page Assets and Workflow

- Tips About Editing Page Assets

- Considerations About Deleting Page Assets

23.4.1 Understanding the Page Asset Model

The page asset model is similar to the flex asset model. This provides an option to change the data structure of the Page asset. The page asset model is made up of the following asset types:

- Page attribute

- Page definition

- Page

The following are some general characteristics of the page asset model:

- Page assets are described by the page attributes that you select for them.

- The page attributes that characterize page assets are themselves assets. This means that attributes can be passed through workflow, edited, monitored by revision tracking, and subjected to all other content management operations.

- If you ever have to add attributes to your asset types in the future (a common occurrence with products), you just create the new attribute and assign it to the appropriate definitions.

- This asset model supports assets that have many, many attributes, which means that you can support large sets of data.

- The page asset model does not have parent definition asset type and parents asset type like the typical flex family, and it does not support data inheritance.

23.4.2 How To Design Page Attributes

Topics:

- For designing attributes, see Designing Flex Attributes.

  Note: Inheritance is not applicable for page assets.

- For designing page definition, see Designing Parent Definition and Flex Definition Assets.

  Note: The Flex Parent Definition is not supported for page assets and inheritance is not applicable.

- For creating page attributes, see Create Flex Attributes in Creating a Flex Asset Family.

- For creating page filter assets, see (Conditional) Creating Flex Filter Assets in Creating a Flex Asset Family.

- For creating page definition assets, see Creating Flex Definition Assets in Creating a Flex Asset Family.

- For creating page associations, see (Conditional) Creating Flex Asset Associations in Creating a Flex Asset Family.

23.4.3 How to Create a Page Asset

To create a page asset:

1. Log in to WebCenter Sites, select the site you want to work with and the icon for the Oracle WebCenter Sites: Contributor interface.

2. Find and bookmark the assets (articles, queries, images, collections, and so on) you want to include on the page. Do the following:

   a. In the **Search** field, enter criteria identifying the asset(s) and then click the **magnifying glass** icon. A search tab opens displaying the results of your search.

   b. In the search results list, select (Ctrl+click) the assets you want to bookmark.

   c. In the search tab's toolbar, click the **Bookmark** icon.

   d. Repeat this step until you have bookmarked all of the assets and then continue with the next step.

   A tab opens displaying the results of your search:

   WebCenter Sites displays a confirmation message and also lists the bookmarked assets under the **Bookmarks** node in the **My Work** tree.

3. In the menu bar, select **Content**, then **New**, and then **Page Asset**. In this example, we use the Page (Home) asset type in the avisports sample site.

   > **Note:**
   >
   > If you are using the avisports sample site, assets of type Page are configured to open in Web Mode. Therefore, when you select to create a Page (Home) asset, the Create Page (Home) dialog box is displayed.

   In the Create Page (Home) dialog box, do the following:

   a. In the **Select Layout** field, select the layout you want to assign to the page.

   b. In the **Name** field, enter a name for the page, and then click **Continue**.

   c. In the avisports sample site the page definition is chosen when you select the type of page asset you will be creating. However, if this is not the case in the site you are working with, you will see a **Page Definition** field. Use this field to specify the page definition for the page you are creating and then click **Continue**.Switch to Form Mode. In the asset's toolbar, click the **Mode** switch. Continue to step 4b.

   d. The asset's Create view is displayed in Form Mode.

4. Create the Page asset.

   a. In the **Name** field, enter a name for the page. type a descriptive name for the page. You can enter up to 64 alphanumeric characters, but the first character must be a letter. Underscores (_) and hyphens (-) are acceptable, but tab and space characters are not.

   b. **Page Definition** field (if applicable): In the avisports sample site the page definition is chosen when you select the type of page asset you will be creating. However, if this is not the case in the site you are working with, you will see a **Page Definition** field. Use this field to specify the page definition for the page you are creating and then click **Continue**.

    c. In the **Template** field, select a template from the drop-down list.

    d. To add items, select the assets from the **Bookmarks** node in the **My Work** tree and then drag and drop the selected items into the field.

5. In the asset's toolbar, click **Save**.

The page is saved. It now displays in the Site Tree under the **Unplaced Pages** pages node.

23.4.4 How To Place Page Assets

After you create a page asset, you position it in the appropriate location in the site tree by using the **Place** function.

1. Open the Admin interface.

2. Click the **Site Navigation** tab, where you should see the site tree with the new page asset in the **Unplaced Pages** list.

3. Expand the site navigation node under which you want to place the page asset.

4. Select a parent for the page you are placing by doing one of the following:

- To place a page at the top-most level in the tree, right-click the site navigation node and select **Place Page** from the context menu.

- Otherwise, right-click the placed page under which you want to insert the new unplaced page, and choose **Place Page** from the context menu.

The place page form opens in the work area on the right. It lists all child pages that are placed under the parent page. It also lists all pages that have not yet been placed in the site tree:

5. To place the page, type a number in the **Rank** field in the list of unplaced pages to designate the new page's position in the list of child page assets. Position numbering starts at 1, at the top of the list.

6. Click **Save**.

The unplaced page asset moves to the site tree, to its assigned rank. To view the page asset in its new location, you may have to right-click in the site tree and choose **Refresh All** from the context menu.

23.4.5 How To Move Page Assets in the Site Tree

In addition to placing unplaced pages, you can also use the place page form to:

- Change the order of child pages within the same parent page.

- Move a child page from one parent page to another.

23.4.5.1 Reordering Child Pages

To re-order children of the same parent page:

1. Open the Admin interface.

2. Click the **Site Navigation** tab and expand the site navigation node containing the pages you want to re-order.

3. Right-click a placed page that has multiple child pages, and choose **Place Page** from the context menu.

   The place page form opens in the work area on the right.

4. In the list of placed child pages, type new values in the **Rank** column to re-order the child pages.

5. Click **Save**.

   The child pages move to their new positions in the site tree.

23.4.5.2 Changing Parent Pages

To move a child page from one parent page to another:

1. Open the Admin interface.

2. In the **Site tree**, expand the **Site Navigation** node, and then expand the site navigation node containing the child page you want to move under a different parent page.

3. Remove the page asset from its parent page:

   a. Right-click the placed page whose child page you want to move, and choose **Place Page** from the context menu.

      The place page form opens in the work area on the right.

   b. In the list of placed child pages, select the **Remove** check box next to the child page that you want to move.

   c. Click **Save**.

      The child page moves to the list of **Unplaced Pages** in the site tree.

4. Place the page asset under its new parent page:

   a. In the site tree, right-click the placed page where you want to insert the unplaced child page, and choose **Place Page** from the context menu.

      The place page form opens in the work area on the right.

   b. In the list of unplaced pages, type a number in the **Rank** field to designate the new page's position in the list of child page assets. Position numbering starts at 1, the top of the list.

   c. Click **Save**.

      The previously unplaced page asset moves under the site navigation node containing the parent asset, in its assigned rank.

23.4.6 Considerations About Placing Page Assets and Workflow

WebCenter Sites has a workflow feature that controls the flow of assets as they pass from one team member to another; for example, from author to editor to approver to publisher. The workflow administrator can create processes that control who can place page assets in the site tree and during which workflow step they can do so. Note the following:

- The **Place Page** workflow privilege controls all place page functions: **Place Pages**, **Remove**, and **Rank**.

- You must have the proper privileges for both the parent page on which you invoke **Place Pages**, and for any child page that you want to **Rank** or **Remove**.

For information about the workflow process, see Creating and Managing Workflow Processes in *Oracle Fusion Middleware Administering Oracle WebCenter Sites*.

23.4.7 Tips About Editing Page Assets

In general, there are two ways to edit an existing page asset:

- Change the assets, but not the asset types, that are included on the page. For example, move new assets to the **Contains** list from your Bookmarks; select a different collection, query, or article from a named association field; or rebuild a collection associated with the page asset to include different assets.

- Create a new association or change the actual structure of the page asset in some way.

Although you may frequently change the content in the collections or queries on a page at regular intervals, you are less likely to change the associations, asset types, or structure of a page after the site goes live. This may also require you to edit the code in the template element that formats the page.

23.4.8 Considerations About Deleting Page Assets

During your site design phase, it is likely that you will create and delete many page assets. However, before deleting a page asset from a site that you have published, be sure that you understand the consequences. For example:

- Have you removed references to the page from other page assets?

- Are any of your other page templates coded to extract and use information about this page asset in any way?

Before you delete a page asset, be sure to remove any references to it from any other elements or pages. It is a good idea to unplace a page asset before you delete it.

24

Best Practices for Creating Future Site Preview Assets and Templates

You can run a special version of a page for an event for as long as the event lasts. In your templates just use the tag that filters assets by date. On the content side, content contributors need to create time-sensitive assets.

Topics:

- About Implementing Future Site Preview
- Creating Sets of Assets
- Writing Templates for Future Site Preview
- Caching Considerations

24.1 About Implementing Future Site Preview

Content contributors can preview in the Oracle WebCenter Sites: Contributor interface how their assets will display on the online site at a future time. You'll use asset start date and end date attributes to enable the Future Site Preview functionality. These two attributes determine the date range during which assets are available on the website. Content creators can specify start dates and end dates in Edit screens in the Contributor interface.

To properly implement the Future Site Preview feature:

- Content contributors must create an appropriate set of assets. See Creating Sets of Assets for more information.

- The administrator must update the templates that render the assets to include the tag `asset:filterassetsbydate`. See Writing Templates for Future Site Preview for more information.

24.2 Creating Sets of Assets

Do you want to display on your site different versions of an item on different dates? You need multiple assets to accomplish your goal. For example, to run a special version of a page for a one-day New Year's Day sale event, you need three assets. A regular page asset, a sales event asset, and a duplicate of the regular page asset.

For example:

- Create the regular page asset, setting its end date to the date before the sale day, December 31st 23:59:59.

- Create the sales event asset, setting the start date to the beginning of the sale day, January 1st 00:00:00, and the end date to the end of the sale day, January 1st 23:59:59.

- Duplicate the regular page asset, setting the start date to the day after the sale, January 2nd 00:00:00.

After the three assets are created, they can be passed as input to the `asset:filterassetsbydate` tag, which will return the asset to render based on the given date.

For ease of use in searching for the related group of assets for editing, publishing, and filtering in templates, we recommend that developers designate an attribute (such as `name`) which content contributors fill in using a naming convention when creating related sets of time-sensitive assets.

24.3 Writing Templates for Future Site Preview

The `asset:filterassetsbydate` tag filters assets according to a given date. Appropriate usage of this tag is critical to Future Site Preview functionality.

To understand the proper use of this tag, see these topics:

- The asset:filterassetsbydate Tag

- The Input List

For more information about the `asset:filterassetsbydate` tag, see the *Oracle Fusion Middleware Tag Reference for Oracle WebCenter Sites Reference*

24.3.1 The asset:filterassetsbydate Tag

The filter tag has the following format:

```
<asset:filterassetsbydate
inputList="inputListName"
outputList="outputListName"
[date="date value in either yyyy-MM-dd HH:mm:ss OR yyyy-MM-dd format"]>
```

Notice that the filter tag has two input attributes (`inputList` and `date`) and one output attribute (`outputList`):

- The `inputList` attribute specifies the list of assets to be filtered based on the given date.

- The `date` attribute is optional. The `date` attribute is expected to be in either `yyyy-MM-dd HH:mm:ss` or `yyyy-MM-dd` format. The `date` attribute should be coded to accept the date value passed from the date picker in the preview screen (use the `__insiteDate` variable for this).

- The tag produces an output list (`outputList`) which contains assets whose start/end dates enclose the given date.

The tag performs the following steps:

- Checks for the `cs.sitepreview` property to determine if the template is stored on a content management system.

- If Future Site Preview is turned off, the system date passes into the date field.

- If Future Site Preview is turned on, the tag routine:

– Disables caching of the current page being rendered (see Caching Considerations for more on caching).

– Accepts the date parameter (passed from the date picker).

– Checks for the appropriate format of the date passed into the tag.

– Filters the input list of assets based on the given date to produce the output list.

24.3.2 The Input List

The `asset:filterassetsbydate` tag requires the input list to contain two columns named `assetid` and `assettype`. This input list can be constructed in a variety of ways.

The simplest way to build the input list, and the way Oracle recommends, is for content contributors to follow a naming convention when filling in the attribute of your choice (usually the `name` attribute). The list-building code could then be written using the `asset:search` tag or other similar tags to search that attribute for the agreed upon string and construct the list from the search results.

Another method for locating assets to place into the input list is to use the Recommendation asset type to hold a list of assets of interest.

Whichever method you use to determine the assets that have to be `filtered`, use the `listobject` tag to construct the input list for the filter tag, as shown in the sample code below.

This sample code creates a list object `inputListName` and adds a row containing two columns: `assetid` and `assettype`. The `listobject:tolist` then creates the input list called `assetInputList`. This list is now ready to be passed as input to the filter tag.

```
<listobject:create name="inputListName" columns="assetid,assettype" />
<listobject:addrow name="inputListName">
<listobject:argument name="assetid" value='<%=ics.GetVar("assetIdVar")%>' />
<listobject:argument name="assettype" value='<%=ics.GetVar("assetTypeVar")%>' />
</listobject:addrow>
<listobject:tolist name="inputListName" listvarname="assetInputList" />
```

Note that for the sake of simplicity, the code snippet explains the creation of an input list containing only one row. In practice users typically have multiple rows (usually read off from the results of `asset:search` tag or some other list) added to the list with each row representing an asset that needs to be filtered by a given date.

After creating the input list of assets to be filtered, use the `asset:filterassetsbydate` tag as follows:

```
<asset:filterassetsbydate inputList="assetInputList" outputList="assetOutputList"
date='<%=ics.GetVar("dateValueVariable")%>' />
```

To pass input from the Future Site Preview date picker to the date attribute, replace the generic `dateValueVariable` with `_insiteDate`.

The tag produces an output list `assetOutputList`. Read through the list for assets that clear the filter by date test as follows:

```
<ics:if condition='<%=ics.GetList("assetOutputList")!=null &&
ics.GetList("assetOutputList").hasData()'%>
<ics:then>
    <ics:listloop listname=assetOutputList>
```

```
      <ics:listget listname=assetOutputList fieldname=assetid output=id />
      <ics:listget listname=assetOutputList fieldname=assettype output=type/>


<!--
Perform your usual asset load, asset get and other rendering functions using
WebCenter
Sites tags here
-->
</ics:listloop>
</ics:then>
</ics:if>
```

24.4 Caching Considerations

Web pages show expected content for the current date only when asset entries are removed from the cache at the proper times. Therefore, WebCenter Sites includes start and end dates in the factors it uses to calculate the expiry time for cached pages.

> **Note:**
>
> WebCenter Sites does not support the use of start and end dates with Export to Disk publishing. When assets are exported to disk, start and end date attributes are also exported to disk. However, the Export to Disk publishing method has no mechanism similar to the cache cleaning process (which, in other publishing methods, automatically removes expired assets from disk).

On a content management system, when a page is previewed, the asset:filterassetsbydate tag disables page caching for that page. This ensures that the page being served always displays assets filtered by the date being passed from the future preview date picker, rather than serving pages from the page cache, which may have been generated using different date input.

25

Configuring Sites for Multilingual Support

How would you empower your site users to translate site pages? Configure multilingual support. You can also create site-specific delivery rules, rules that determine the language versions of assets to be shown on the online site, and also handle visitors' requests for unavailable language versions.

When you configure a site for multilingual support, users in that site gain the ability to assign *locale* (language version) designations to assets, and to create translations of assets.

Topics:

- About Configuring a Site for Multilingual Support

- Working with Locale Filtering

- Planning Multilingual Support for a Site

- Configuring Multilingual Support for a Site

- Tips for Using WebCenter Sites Translation Mechanism

25.1 About Configuring a Site for Multilingual Support

Are you looking for answers to questions such as these: what's the best way to differentiate semantically identical assets and group locales, how you should use locales and dimension sets for cross-site multilingual support, what's the best way to handle asset relationships and approval dependencies during editing and translation?

See these topics:

- Dimensions

- Dimension Sets

- Cross-Site Multilingual Support

- Master Assets, Translations, and Multilingual Sets

- Translations and Asset Relationships

- Approval Dependencies

25.1.1 Dimensions

Locale designations in WebCenter Sites are implemented through the concept of *dimensions*. A dimension is an identifier that differentiates assets that are otherwise semantically identical. A locale (such as en_US for US English) is thus a type of dimension that differentiates two translations of the same content.

Dimensions are represented by assets of type Dimension. This asset type must be enabled by developers on a per-site basis so that users can create Dimension assets (of subtype Locale), and so that content providers can assign locale designations to assets they want to translate.

> **Note:**
>
> Users cannot create translations of assets that have no locale designation assigned.

Each Dimension asset represents a locale in the site. For example, an `en_US` Dimension asset represents US English, and an `fr_CA` Dimension asset represents Canadian French. When a content asset is assigned a locale, the assignment is recorded in the `assetType_Dim` table for the corresponding asset type.

Publishing content in a given locale requires enabling the locale on the online site, that is, publishing the Dimension asset representing the locale to the delivery system, and including the locale in the site's dimension set.

25.1.2 Dimension Sets

When you have created your locales, we recommend that you create at least one dimension set. A dimension set is a grouping of dimension assets (locales), which affects the delivery of content to the site visitor in the following ways:

- A dimension set defines which locales are designated as enabled for the online site. In other words, a dimension set determines the languages in which content will be shown to the visitors. For example, one dimension set would contain European languages, another Asian languages, and so on.

- A dimension set defines how to filter content based on locale. For example, how to handle content that does not exist in the visitor's preferred language at the time the visitor requests it. If you do not publish a dimension set to the delivery system, locale filtering will not function. For information on locale filtering, see Working with Locale Filtering.

> **Note:**
>
> Dimension sets do not affect the operation of WebCenter Sites user interfaces.

For locale filtering to function on the online site, you must approve and publish to the delivery system both the DimensionSet assets and the Dimension assets referenced by each dimension set. (Because referenced Dimension assets are dependents of the DimensionSet assets referencing them, they must be approved with their respective DimensionSet assets.)

25.1.3 Cross-Site Multilingual Support

If you are setting up multilingual support in multiple sites, you can choose to implement one of the following scenarios:

- **Create the locales, but no dimension sets**

This option allows content providers to manage content in multiple languages, but does not enable render-time locale filtering. Use this option only if translations in all required languages will exist for each asset from the very beginning.

- **Create the locales and a dimension set, and share them across your sites**

    This option provides the simplest way of enabling multilingual support on multiple sites. Sites set up in this way share the properties stored in the dimension set (locales enabled for display on the online site, the locale filtering method, and, if applicable, the fallback hierarchy (the path the Hierarchical filter traverses when looking up asset translations at render time). If you are creating a bare-bones site that you will replicate into multiple target sites, it is best to share your locales to the target sites.

- **Create separate locales and dimension sets for each site**

    This option affords the most flexibility, at the cost of increased configuration complexity. Sites set up this way benefit from the fact that properties such as locale filter type or fallback hierarchy can be tailored to each site.

    > **Note:**
    >
    > While creating duplicate Dimension assets to represent the same language in multiple sites is possible, it is not recommended, as it introduces unnecessary complexity.

- **A mixture of the latter two options**

    This option provides the right balance between flexibility and configuration complexity. As a possible best practice, you would create a pool of unique locales, share the locales required by each site from that pool, and share or create dimension sets for each site as needed.

25.1.4 Master Assets, Translations, and Multilingual Sets

When an asset is assigned a locale for the first time, it gains **master**, or **dimension parent**, status. Master status allows the formation of a multilingual set (a group of assets whose content is semantically identical, but exists in different languages. (Note that this is not the same as a dimension set; a dimension set only affects the online site and the way assets are rendered.)

> **Note:**
>
> The terms master asset and dimension parent are equivalent. Master asset is displayed in WebCenter Sites's user interfaces; dimension parent is used in the *Oracle Fusion Middleware Tag Reference for Oracle WebCenter Sites Reference*, database table names (such as *assetType*_DimP), and element code.

For example, when you designate an Article asset as US English (en_US), and create translations of it in whichever locales are enabled in the site (such as French (fr_CA) and German (de_DE)), the translations point to their dimension parent (the US English asset) to indicate they are semantically equivalent to the master and one another.

When you create a translation of an asset with master status, WebCenter Sites copies the asset and assigns the locale of your choice to the copy. You then enter the translated content and save the translation as a new asset.

At this point, the source asset and its translation are linked into a multilingual set, and the translation adopts the source asset as its master, or dimension parent. Any member of the set that is not the master can be given master status; however, only one set member at a time can be the master.

The linking is accomplished through the *assetType*_DimP table for the asset type. The table stores the following information:

- ID of the master (dimension parent) asset

- ID of the translation asset

- ID of the locale dimension asset assigned to that translation

Even though WebCenter Sites interfaces allow you to initiate the creation of a translation from either the master asset or any of its existing translations, all translations in the multilingual set always point to the dimension parent (master) asset.

> **Note:**
>
> If a locale-aware asset is being revision-tracked, changes to asset locale data (such as locale designation or master status) do not generate a new version of the asset.

25.1.5 Translations and Asset Relationships

The way asset relationships are handled when an asset is translated is summarized in this table:

Table 25-1   Asset Relationships

Relationship Type	Behavior
Associations	When an asset containing associations is translated, all assets associated with the source asset are automatically associated with the translation. You then have the choice to translate the associated assets and associate the translated versions with the translated parent asset.
Collections	When you create a translation of a Collection asset, the new Collection asset retains the member assets of the source asset. You then have the choice to translate the member assets and place the translated versions in the new Collection asset, replacing the member assets carried over from the old collection.
Static Lists Recommendations	When you create a new language version of a Static Lists recommendation, the new Recommendation asset retains the member assets of the source asset. You then have the choice to translate the member assets and place the translated versions in the new Recommendation asset, replacing the member assets carried over from the old collection.
Dynamic Lists Recommendations	Since Dynamic Lists recommendations are populated by element code, they are not affected.

Table 25-1 (Cont.) Asset Relationships

Relationship Type	Behavior
Related Items Recommendations	When an asset containing Related Items associations is translated, all assets associated with the source asset are automatically associated with the translation. You then have the choice to translate the associated assets and associate the translated versions with the translated parent asset.
Asset-Type Attributes	When an asset containing associations through asset-type attributes is translated, all assets associated with the source asset are automatically associated with the translation. You then have the choice to translate the associated assets and associate the translated versions with the translated parent asset.
Embedded Links	Embedded links are not affected. When an asset containing embedded links is translated, you must manually update the links to point to the corresponding translations of the linked content (if such translations exist).

For information on handling asset relationships at render time, see Working with Locale Filtering.

25.1.6 Approval Dependencies

An approval dependency exists between two assets when editing one of the assets causes the other's approval status to change. This table summarizes the approval dependencies affecting localized assets:

Table 25-2 Approval Dependencies

Dependency	Effect on Asset Approval
An Exists dependency exists between a localized asset and the Dimension asset representing the assigned locale.	To approve a localized asset for publishing, the corresponding Dimension asset must also be approved.
In a multilingual set, an Exists dependency exists between the master asset and each translation linked to it.	When you create the first translation of an asset, you must approve both the asset and its translation. To approve a translation, you must also approve the corresponding master asset, unless the master asset has been approved. You must reapprove all members of the set if: • You add a new translation to, or delete an existing translation from the set. • You edit the set's master asset. • You designate another member of the set as the master.
An Exists dependency exists between a DimensionSet asset and the Dimension assets representing the locales enabled in that dimension set.	To approve a dimension set, the corresponding Dimension assets must also be approved.

25.2 Working with Locale Filtering

On the online site, visitors may choose a language in which the assets of their interest have not yet been translated. You can use a locale filter that decides which translation should show on the site in the absence of the visitor's preferred language and in the current circumstances.

For example, you could let the business logic decide what to do if a requested asset does not exist in the requested language.

By using locale filtering, you can also spread editorial work over time by allowing content providers to create the required translations after the original content is published to the online site. Locale filtering allows the site to automatically pick up the missing translations as soon as they are published to the delivery system.

Keep in mind that locale filtering introduces additional load on the delivery system. The amount of additional load depends on the complexity of the filtering logic.

This section includes the following topics:

- Options for Implementing Asset Relationships Through Locale Filtering

- Understanding the Included Locale Filters

- About Using Custom Locale Filters

- Accounting for Compositional Dependencies

- About Adding Filtering Support to Your Site

25.2.1 Options for Implementing Asset Relationships Through Locale Filtering

The way you choose to implement locale filtering will have an influence on how asset relationships on your site are structured, and vice versa, depending on the way you want the online site to behave.

You can choose to implement one of the following options:

- **Maintain different asset relationship trees for each locale**

  When rendering assets, this model renders whatever assets are associated with the requested asset.

  For example, if an asset exists in English and French, and each version has a unique set of associated assets, each version is rendered with its respective associated assets. Filtering is only used to look up and deliver a version of the requested asset matching the language preference specified by the visitor; the associated assets are expected to have been translated into all required languages.

  This model allows for completely independent content for each language. It is used in the First Site II sample site.

- **Use the same asset relationship tree for all locales**

  When rendering assets, this model substitutes the associated assets of the requested asset with the assets associated with a specific language version of the requested asset, regardless of the language preference specified by the visitor.

  For example, if an asset exists in English and French, each version has a unique set of associated assets, and the visitor specified French as their language preference, filtering will look up and deliver the French version of the requested asset, but it

will substitute the associated assets of the English version in place of those of the French version (assuming the language version from which filtering is to derive associations is English).

This model ensures consistent content across all languages.

- **A mixture of the two models**

  Allows for the greatest amount of flexibility and customization for your site. The optimal proportion between the two models will depend on the intended behavior of your site.

25.2.2 Understanding the Included Locale Filters

WebCenter Sites ships with the following locale filters:

- The Simple Filter

- The SimpleLookup Filter

- The Hierarchical Filter (also known as the Fallback filter)

You also have the option to implement custom locale filters, if desired. For information about custom filters, see About Using Custom Locale Filters.

Note that depending on the type of filter you choose to implement, the assets being filtered must satisfy one, or both of the following conditions:

- Assets must have locale designations assigned. Assets without locale designations will be ignored by locale filters.

- Assets that are translations of one another must be linked into multilingual sets (that is, designated as translations of one another through a master asset). Otherwise, the filters are not able to perform the necessary translation lookups.

25.2.2.1 The Simple Filter

The Simple filter is a possible choice for a site that should only be rendered in one language, but whose content exists in multiple languages. The filter checks the following:

- Whether the requested asset is in the language specified by the visitor

- Whether the locale of the asset is listed in the site's dimension set

If both conditions are met, the filter passes the asset to the template for rendering; otherwise, nothing is rendered.

The Simple filter has the least impact on delivery system performance, but increases the amount of editorial work that needs to be done, as assets must exist in the required language versions or they are not displayed on the online site.

25.2.2.2 The SimpleLookup Filter

The SimpleLookup filter is ideal for a site that should only be rendered in one language, but whose content may exist in multiple languages, and for which there is no guarantee that all of the necessary translations exist at render time. The filter checks the following:

- Whether the requested asset is in the language specified by the visitor

- Whether the locale of the asset is listed in the site's dimension set

If the requested asset is not in the visitor's preferred language, the filter looks up a suitable replacement by checking the asset's translations. If the filter finds a matching translation, it passes it to the template; otherwise, nothing is rendered. (The filter will also return nothing if the locale of the translation is not included in the site's dimension set.)

This filter offers a reasonable balance between performance and functionality. While the lookup queries slightly increase the load on the delivery system, the amount of editorial work done to create assets can be reduced, as the required translations can be created after the original content is published to the online site. The lookup mechanism will pick up the missing translations as soon as they are published to the delivery system.

25.2.2.3 The Hierarchical Filter

The Hierarchical filter checks whether the locale of the requested asset matches the locale requested by the visitor. If the locales do not match, the filter checks the asset's translations to see if a suitable replacement exists. If the filter finds a matching translation, it passes it to the template; otherwise, it substitutes translations of the requested asset according to the fallback hierarchy you set up when you configure the site's dimension set. The fallback hierarchy determines which language versions the filter should substitute for the requested asset, and in what order.

For example, consider the following hierarchy:

- **en_US** (US English)
- **de_DE** (German)
  - **de_CH** (Swiss German)
  - **de_AT** (Austrian German)
- **fr_FR** (French)
  - **fr_BE** (Belgian French)
  - **fr_CA** (Canadian French)
- **en_UK** (British English)

In our example, when the visitor requests an asset in Swiss German (de_CH), the filter looks up the asset's translations and if it finds a Swiss German version of the asset, it passes that version to the template. If the filter cannot find a Swiss German version, it falls back to the next best locale in the hierarchy path, German (de_DE). If, in turn, no German translation exists, the filter follows the path specified in the hierarchy until it reaches the top of the tree. If no match is found in the process, nothing is rendered.

Note that the above example describes a situation in which the visitor specifies a single preferred language. If the user specifies multiple preferred languages (in most cases, in the form of an ordered list), the filter attempts to find a match in the fallback hierarchy for the visitor's most preferred language. If no match is found, the filter checks the next language on the visitor's list, until a match in the fallback hierarchy is found. When that happens, the filter attempts to substitute translations of the requested asset by tracing a path from the matching locale to the top of the fallback tree, as described earlier.

For example, if the user's preferred languages are Japanese, French, and English (in that order), the filter attempts to locate Japanese in its fallback hierarchy. Since Japanese is not in the hierarchy, the filter then attempts to locate French. French is in the hierarchy, so the filter traces a path from French to the root node of the tree, and attempts substitution according to that path, as illustrated by the example earlier in this section.

While powerful and convenient, the hierarchical filter has the following drawbacks:

- The additional database queries run by the filter tax the performance of the delivery system. To minimize the performance hit, editorial work should be done to ensure that content exist in as many of the required languages as possible, so that the filter's activity is minimized. (You may also choose to use a different filter.)

- Control over which assets to display on the online site is put exclusively in the hands of the site developer or administrator. This is because the filter follows the fallback tree configured in the dimension set, rather than the preference order specified by the site visitor (assuming the site is set up to accept multiple language preferences from each visitor).

25.2.3 About Using Custom Locale Filters

Depending on the design of your site, you may decide to create custom filters. For example, your site design might call for a hierarchical (fallback) filter that favors the locale priority specified by the visitor, rather than the one defined in the dimension set. In such cases, a field in the Edit form for the DimensionSet asset lets you specify a custom filter class.

25.2.4 Accounting for Compositional Dependencies

If you decide to incorporate locale filtering on your site, you must account for the additional compositional dependencies that are introduced as a result. Compositional dependencies determine how pages are cached on your delivery system.

This section includes the following topics:

- Asset Lookup Chain

- Caching Rules

25.2.4.1 Asset Lookup Chain

When using locale filtering to look up a translation of an asset, the following factors determine how pages are cached, based on which assets are loaded during the lookup process:

- The filtering logic employed

- The page and asset from which the lookup request originates

- The language preference specified by the visitor

A cached page containing the requested asset is dependent on all assets loaded during the lookup process. Thus, if an asset that is loaded during the lookup process is modified, the affected page is flushed from the cache.

For example, consider the SimpleLookup filter and the following multilingual set:

- **en_US** (master)

- **fr_FR** (translation)

- **de_DE** (translation)

If a visitor requests a page containing the French version, but the visitor's language preference is German, the lookup chain is as follows:

fr_FR > en_US > de_DE

In this example, all three assets are loaded, because the filter must first load the master asset linked to the French version, and then use the master asset to look up the German version. Thus, if any of these three assets is modified, the affected page is flushed from the cache.

If, on the other hand, the user requested the US English version, which is the master asset of the set, then the lookup chain would be shorter:

fr_FR > en_US

In such case, the French and US English versions are loaded, but the German version is not. Thus, modifying the German version would not cause the corresponding page to be flushed from the cache, but modifying the French or US English versions would.

For a detailed explanation of the lookup mechanisms employed by locale filters included with WebCenter Sites, see Understanding the Included Locale Filters.

Caching Rules explains the caching rules applicable to multilingual assets.

25.2.4.2 Caching Rules

Once the translation lookup occurs and the affected page is cached, the page is flushed from the cache whenever one of the following occurs:

- A new language is added to the multilingual set

- A translation that was part of the lookup chain when the page was rendered is edited

- A translation that is a member of the multilingual set is deleted

- The set's master asset is edited

- Another member of the set is designated as the master

25.2.5 About Adding Filtering Support to Your Site

To add support for locale filtering to your site, you must modify the templates and element code used on your site.

When the template code fetches an asset via the asset's `c/cid` values, the locale filter executes its business logic on the incoming `c/cid` values and returns the resulting `c/cid` values (or nothing) to the template for rendering.

The structure of your site will influence how you implement locale filtering in your templates, and vice versa. It will also determine the behavior of your site in different scenarios.

For example, imagine five articles, each existing in two languages, `en` (English), and `fr` (French). The articles would be `a1en`, `a1fr`, `a2en`, `a2fr`, and so on. We can decide to put these articles into a collection and implement locale filtering in one of the following ways:

- Create an English collection, `clen`, and assign all of the English articles to it. This way, before we render the collection, we would simply filter the `c/cid` of the `clen` asset, then render its children without filtering their `c/cid` values, because we trust the `clen` collection to be in a single language.

- Create a multilingual collection (without assigning a locale to it) and add the articles in whatever languages are desired. Then, when rendering each article, filter the article `c/cid` values so that the article is rendered in the locale specified by the visitor.

The rest of this section provides code examples based on the FirstSiteII sample site. We recommend that you examine the FirstSiteII code to get an idea of how it multilingual support.

This section includes the following topics:

- About Adding Filtering to Templates

- About Obtaining and Maintaining a Visitor's Locale Preference

- About Filtering Search Results

25.2.5.1 About Adding Filtering to Templates

Usually, you would place your filter code into a utility element and call the element at the top of the template to process the `c/cid` values.

The following example shows how the `FSIILayout` template calls the filter code stored in the `FSIICommon/Multilingual/Filter` element asset via the `render:lookup` tag:

```
<%-- Execute the Dimension filter to look up the translated asset that corresponds
to the locale that the visitor requested. --%>
<render:lookup site='<%=ics.GetVar("site")%>' varname="Filter" key="Filter"
match=":x" tid='<%=ics.GetVar("tid")%>' />
<render:callelement elementname='<%=ics.GetVar("Filter")%>' scoped="global"/>
```

25.2.5.2 About Obtaining and Maintaining a Visitor's Locale Preference

For filtering to work, you have to allow the visitor to specify a preferred language (locale). This preference must then be propagated throughout the entire site (that is, passed to all the templates).

The example below shows how this is accomplished in the `FSIIWrapper` element. The last section of this example shows how the locale variable is set by taking the value from the session variable (earlier in the example, we ensure that the session variable exists).

```
<%-- The session variable locale refers to the id of the dimension with the
subtype of Locale that specifies which language the site is to be rendered in.
Users can select the locale of their choice from a menu on every page of the
site, and once selected, it is stored in session. A default locale is mapped to
this CSElement and is set if it has not already been set. --%>
<ics:if condition='<%=ics.GetSSVar("preferred_locale") == null%>'>
<ics:then>
<render:lookup site='<%=ics.GetVar("site")%>' varname="default:locale:name"
key='DefaultLocale' ttype="CSElement" tid='<%=ics.GetVar("eid")%>' match=":x"/>
<asset:load name="defaultLocale" type="Dimension" field="name" value='<
%=ics.GetVar("default:locale:name")%>'/>
<asset:get name="defaultLocale" field="id" output="default:locale:id"/>
<ics:setssvar name="preferred_locale" value='<%=ics.GetVar("default:locale:id")%>'/>
</ics:then>
```

```
</ics:if>
<%-- Call the wrapped child page. There is no need to look up the template or to
enable any special PageBuilder functionality, so we can use the render:satellitepage
tag in this situation. --%>
<render:satellitepage pagename='<%=ics.GetVar("childpagename")%>' packedargs='<
%=ics.GetVar("packedargs")%>'>
<render:argument name='c' value='<%=ics.GetVar("c")%>'/>
<render:argument name='cid' value='<%=ics.GetVar("cid")%>'/>
<render:argument name='p' value='<%=ics.GetVar("p")%>' />
<render:argument name="locale" value='<%=ics.GetSSVar("preferred_locale")%>'/>
</render:satellitepage>
```

25.2.5.3 About Filtering Search Results

If your online site contains search functionality, you may choose to filter the search results returned to the visitor, based on the visitor's language preference.

The following example shows how the `Page/SearchDetailView` template filters an `IList` of search results so that the query can go against all languages but only return defined results:

```
<%-- look up the dimension set and filter the ProductList results --%>
<asset:load name="GlobalDimSet" type="DimensionSet" field="name" value='<
%=ics.GetVar("GlobalDimSet")%>' />
<dimensionset:filter name="GlobalDimSet" tofilter="ProductList" list="ProductList">
<dimensionset:asset assettype="Dimension" assetid='<%=ics.GetVar("locale")%>'/>
</dimensionset:filter>
```

See the *Oracle Fusion Middleware Tag Reference for Oracle WebCenter Sites Reference*. Additionally, examine the code in the FirstSiteII sample site to see how it implements multilingual support.

25.3 Planning Multilingual Support for a Site

Before you start configuring multilingual support on your site, consider how many languages you need to implement at present, whether you would like to share locale and dimension sets between sites, consider how you should handle asset relationships, and so on. Make these decisions in agreement with your site administrators.

1. Determine how many languages to initially implement on your site (or sites), based on your organization's content management needs. You can either:

   - Build a site (or sites) to support a single language initially, and add support for additional languages as the need arises.

   - Plan ahead for all the languages you expect to incorporate across all of your sites and create the appropriate locales in advance.

2. Determine whether you will share existing locales and dimension sets to the new site (or sites), or create separate ones. See Cross-Site Multilingual Support.

3. Decide how asset relationships are going to be handled at render time with respect to locales, and choose the locale filtering method(s) appropriate to your decision. The choices you make will have to strike a balance between levels of automation, delivery system performance, and editorial workload, and are thus best made in agreement with your site administrators. See Working with Locale Filtering. Note the following:

   - Different filtering methods provide different levels of automation at the cost of a performance hit on the delivery system. The more complex the filter, the

higher the performance hit. For example, the SimpleLookup filter provides better performance than the Hierarchical filter.

- Depending on the filtering method you implement, editorial work on site content can be spread over time, as translations can be created after the original content is created and published to the live site. The filtering logic you implement will decide what to do content that does not yet exist in the required language versions.

4. If you are converting a monolingual site to a multilingual site, obtain the element code you will use to assign the default locale to the assets in the site. Sample code based on the FirstSiteII sample site is provided in Sample Element Code for Bulk-Assigning a Default Locale.

Note:

When replicating a site containing multilingual sets, make sure the master assets are available on the target site (by either sharing or copying). Otherwise, the set members will no longer be linked as translations of each other on the target site.

25.4 Configuring Multilingual Support for a Site

First thing you do is enable Dimension and DimensionSet asset types. And then you begin creating dimension sets and locales which you may share with other sites if you've planned to do so. When you're configuring multilingual support, you also configure a fallback hierarchy.

These topics describe the procedures necessary to configure a site for multilingual support:

- Configuration Quick Reference

- Enabling the Dimension and DimensionSet Asset Types

- Enabling the Locale Subtype of the Dimension Asset Type

- How To Create a Locale

- How to Share a Locale to Another Site

- How To Create and Configure a Dimension Set

- How To Share a Dimension Set to Another Site

- How To Configure a Locale Filter

- How to Configure the Fallback Hierarchy of the Hierarchical Filter

- How to Bulk-Assign a Default Locale to Assets in a Site

25.4.1 Configuration Quick Reference

This section provides an overview of the steps necessary to configure multilingual support for a site. Use this list as a quick reference during the configuration process.

1. Make the necessary decisions and preparations as described in Planning Multilingual Support for a Site.

2. Enable the Dimension and DimensionSet asset types on the site. For instructions, see Enabling the Dimension and DimensionSet Asset Types.

3. Enable the Locale subtype of the Dimension asset type on the site. For instructions, see Enabling the Locale Subtype of the Dimension Asset Type.

4. Create or share locales. For instructions, see the following sections:

- For creating new locales, see How To Create a Locale.

- For sharing existing locales, see How to Share a Locale to Another Site.

For help in determining whether to create locales or share existing ones, see Cross-Site Multilingual Support.

5. Create or share a dimension set. For instructions, see the following sections:

- For creating a new dimension set, see How To Create and Configure a Dimension Set.

- For sharing an existing dimension set, see How To Share a Dimension Set to Another Site.

For help in determining whether to create a new dimension set or share an existing one, see Cross-Site Multilingual Support.

6. (Optional) If you are converting an existing monolingual site to a multilingual site, execute element code that assigns a default locale to each asset in the site. For instructions, see How to Bulk-Assign a Default Locale to Assets in a Site. The section includes sample code which you can customize for your site.

7. Modify the templates used in the site to include support for the locale filter you selected when you configured the dimension set. For an overview of the process, see About Adding Filtering Support to Your Site.

Note:

In addition to locale filtering, you will have to implement the following site functionality:

- Allow the visitor to specify their language preference.

- Propagate the visitor's language preference throughout the site (by passing it to all templates on the site).

- Maintain the visitor's language preference for the duration of the session.

25.4.2 Enabling the Dimension and DimensionSet Asset Types

Before you can create Dimension and DimensionSet assets in your site, you must enable the corresponding asset types and subtypes. This procedure describes how to enable the Dimension and DimensionSet asset types on your site. The following procedure describes how to enable the Locale subtype of the Dimension asset type on your site.

1. Open the Admin interface.

2. Under the **General Admin** tree, expand the **Admin** node.

3. Under the **Admin** node, drill down the following hierarchy:

   a. Expand the **Sites** node.

   b. Under the **Sites** node, expand the node corresponding to the site.

   c. Under the site node, expand the **Asset Types** node.

   d. Under the **Asset Types** node, double-click the **Enable** node.

   WebCenter Sites displays the Enable Asset Types form.

4. In the Enable Asset Types form, select the check boxes next to the **Dimension** and **DimensionSet** asset types.

5. Click **Enable Asset Types**.

   WebCenter Sites displays the **Start Menu** Selection form.

6. In the **Start Menu** Selection form, select all of the check boxes, and click **Enable Asset Types**.

   WebCenter Sites displays a message confirming the asset types have been enabled for the site.

25.4.3 Enabling the Locale Subtype of the Dimension Asset Type

Before you can assign locales to assets in your site, you must enable the Locale subtype of the Dimension asset type on your site.

1. Open the Admin interface.

2. Under the **General Admin** tree, expand the **Admin** node.

3. Under the **Admin** node, drill down the following hierarchy:

   a. Expand the **Asset Types** node.

   b. Under the **Asset Types** node, expand the **Dimension** node.

   c. Under the **Dimension** node, double-click the **Subtypes** node.

   WebCenter Sites displays the Subtypes for Asset Type: Dimension form.

4. In the form, click the **Edit** (pencil) icon next to the **Locale** subtype.

   WebCenter Sites displays the Edit Dimension Subtype: Locale form.

5. In the **Sites** field, select **Ctrl+click** for the site to enable the Locale subtype.

   > **Note:**
   >
   > You must select **Ctrl+click** the name of your site to keep the existing site selections intact; if you simply click the site name, other selected sites (if any) will be deselected.

6. Click **Save**.

25.4.4 How To Create a Locale

> **Note:**
>
> If the locale you want to create designates a language that is represented by a Dimension asset in another site in your WebCenter Sites system, share the existing Dimension asset representing that language to your current site instead to avoid redundancy.

To add a new locale to your site, create a Dimension asset of subtype Locale representing a locale, by performing the following steps:

1. In the button bar, click **New**.

2. In the list of asset types, click **New Dimension**.

3. WebCenter Sites displays the New Dimension form.

4. In the New Dimension form, do the following:

   In the **Name** field, enter a descriptive name for the locale. It is recommended that you use the following convention as a best practice:

   xx_YY

   where:

   - xx is the two-letter ISO 639-1 language code (for example, fr for French)

   - YY is the two-letter ISO country code (for example, CA for Canada)

   To complete the above example, the name fr_CA would denote Canadian French.

5. In the **Description** field, enter a description of the language this locale represents.

6. In the **Subtype** drop-down list, select **Locale**.

7. Click **Save**.

25.4.5 How to Share a Locale to Another Site

To share a locale to another site, you must share the corresponding Dimension asset.

1. Open the Admin interface and select the site containing the Dimension assets for the locales you want to share.

2. Find the Dimension asset and open its Inspect form:

   a. In the button bar, click **Search**.

   b. In the list of asset types, click **Find Dimension**.

   c. Enter search criteria (if any), and click **Search**.

   d. In the list of search results, navigate to the asset and click its name.

      WebCenter Sites opens the asset in the Inspect form.

3. In the action bar, select **Share Dimension**.

WebCenter Sites displays the Share Dimension form.

4. In the Share Dimension form, select the check boxes next to the sites to which you want to share the Dimension asset. (To share the asset to all sites on your WebCenter Sites system, select the **All Sites** check box.)

5. Click **Save Changes**.

WebCenter Sites displays a message confirming the asset is now available in the sites you selected.

25.4.6 How To Create and Configure a Dimension Set

To create and configure a Dimension set:

1. Add the Dimension assets (locales) you want to include in the dimension set to your Bookmarks by doing the following:

a. In the button bar, click **Search**.

b. In the Search form, click **Find Dimension**.

c. Enter search criteria (if any) and click **Search**.

d. In the list of search results, navigate to the **Dimension** assets and select their check boxes.

e. Click **Add to My Bookmarks**.

2. Create and configure the dimension set by doing the following:

a. In the button bar, click **New**.

b. In the New asset list, click **New DimensionSet**.

WebCenter Sites displays the New DimensionSet form.

c. In the **Name** field, enter a descriptive name for the dimension set.

d. In the tree, select the **Bookmarks** tab.

e. In the **Bookmarks** tab, select a locale you want to add to the dimension set and click **Add Selected Items**. Repeat this step for each additional locale you want to add.

f. In the **Dimension Filter Class** field, select the locale filter type. The **Advanced** option lets you specify a custom filter class.

g. (Optional) If you selected **Advanced** in step 2, enter the name of the custom filter class into the text box that opens.

h. When you are finished, click **Save Changes**.

i. (Optional) If you selected the **Hierarchical** filter in step 2, complete the steps in How to Configure the Fallback Hierarchy of the Hierarchical Filter to configure the filter's fallback hierarchy.

25.4.7 How To Share a Dimension Set to Another Site

To share a dimension set to another site, you must share the corresponding DimensionSet asset.

1. Open the Admin interface and select the site containing the DimensionSet asset you want to share.

2. Find the DimensionSet asset and open its Inspect form:

   a. In the button bar, click **Search**.

   b. In the list of asset types, click **Find DimensionSet**.

   c. Enter search criteria (if any), and click **Search**.

   d. In the list of search results, navigate to the asset and click its name.

      WebCenter Sites opens the Inspect form for the asset.

3. In the action bar, select **Share DimensionSet**.

   WebCenter Sites displays the Share DimensionSet form.

4. In the Share DimensionSet form, select the check boxes next to the sites to which you want to share the DimensionSet asset. (To share the asset to all sites on your WebCenter Sites system, select the **All Sites** check box.)

5. Click **Save Changes**.

6. WebCenter Sites displays a message confirming the asset is now available in the sites you selected.

25.4.8 How To Configure a Locale Filter

Usually, you configure the locale filter when you create the dimension set for your site. To make changes to the locale filter configuration in an existing dimension set, do the following:

1. Find the dimension set whose locale filter you want to configure and open in the Inspect form:

   a. In the button bar, click **Search**.

   b. In the Search form, click **Find DimensionSet**.

   c. Enter search criteria (if any) and click **Search**.

   d. In the list of search results, navigate to the asset and click its name.

      WebCenter Sites opens the asset in the Inspect form.

2. In the **Dimension Filter Class** field, select the option next to the filter type. The **Advanced** option lets you specify a custom filter class.

3. (Optional) If you selected **Advanced** in step 2, enter the name of the custom filter class into the text field that displays.

4. Click **Save Changes**.

5. (Optional) If you selected the **Hierarchical** filter in step 2, complete the steps in How to Configure the Fallback Hierarchy of the Hierarchical Filter to configure the filter's fallback hierarchy.

25.4.9 How to Configure the Fallback Hierarchy of the Hierarchical Filter

If you selected the Hierarchical (Fallback) locale filter when configuring your dimension set, perform the following steps to configure the filter's fallback hierarchy:

1. If you plan to add new locales or rearrange existing locales in the fallback hierarchy, add the Dimension assets representing the locales to be included in the hierarchy to your Bookmarks by doing the following:

    a. In the button bar, click **Search**.

    b. In the Search form, click **Find Dimension**.

    c. Enter search criteria (if any) and click **Search**.

    d. In the list of search results, navigate to the Dimension assets and select their check boxes.

    e. Click **Add to My Bookmarks**.

2. Find and open in the Inspect form the dimension set that contains the Hierarchical filter you want to configure:

    a. In the button bar, click **Search**.

    b. In the Search form, click **Find DimensionSet**.

    c. Enter search criteria (if any) and click **Search**.

    d. In the list of search results, navigate to the DimensionSet asset and click its hyperlinked name.

    WebCenter Sites displays the Inspect form for the DimensionSet asset.

3. When configuring the fallback hierarchy, note the following:

    • A locale can appear in the fallback hierarchy only once.

    • If you have to delete a locale from the hierarchy, click the **Delete** (trash can) icon next to the locale node.

    • If you have to change the position of a locale in the hierarchy, delete it, then add it under the parent node.

To configure the locale hierarchy:

1. In the **Dimension Filter Class** field, click **Configure Locale Hierarchy**.

WebCenter Sites displays the Configure Locale Hierarchy form.

2. In the Configure Locale Hierarchy form, click **Edit**.

WebCenter Sites displays an editable version of the form.

3. In the tree, select the **Bookmarks** tab.

4. (Optional) If the hierarchy is empty, select in the **Bookmarks** tab the locale you want to designate as the top node of the fallback hierarchy, then click **Add Selected Items**.

5. Select a parent node for the locale you want to add to the hierarchy.

   If you are building a hierarchy from scratch, your only choice will be the top-level node you added in step 4.

   When building your hierarchy, keep in mind the direction in which the fallback process occurs (from most-specific to least-specific; that is, towards the root node of the tree).

6. In the **Bookmarks** tab, select the locale you want to appear under the parent node you selected in step 5, then click **Add Selected Items**.

7. Repeat steps 5 and 6 for each additional locale you want to add to the hierarchy.

8. When your fallback hierarchy is complete, click **Save Changes**.

25.4.10 How to Bulk-Assign a Default Locale to Assets in a Site

If you are converting a monolingual site to a multilingual site, you must assign a default locale to all assets in the site. The fastest way to accomplish this is to execute an element that assigns the default locale to the assets.

For your convenience, sample element code for this procedure is provided Sample Element Code for Bulk-Assigning a Default Locale. The sample code is intended as an example, and will have to be customized for your site.

1. Create a CSElement asset to hold the element code that will assign a default locale to your assets.

2. Create a SiteEntry asset that references the CSElement asset you created in step 1.

3. Call the SiteEntry asset you created in step 2 in a URL, as follows:

   ```
   http://<host>:<port>/<context>/ContentServer?pagename=
   <siteentry_name>
   ```

   where:

   - `<host>` is the host of your WebCenter Sites system

   - `<port>` is the port number on which WebCenter Sites is listening for connections

   - `<context>` is the application context root assigned to the WebCenter Sites application.

   - `<siteentry_name>` is the name of the SiteEntry asset you created in step 2.

   When the element code completes execution, check to ensure your assets have the locale assigned. If not, check the element code for possible errors.

25.4.10.1 Sample Element Code for Bulk-Assigning a Default Locale

This section contains sample element code written for the FirstSiteII sample site. The code does the following:

1. Creates a Dimension asset named `en_US` to represent your default locale designation within the site (US English in this example).

2. Assigns this default locale to all Page assets within the site.

Note:

The code in this section is provided as an example. If you decide to use it, be sure to customize it for your site. Test the code before deploying it; no error checking is included in this example.

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"%>
<%@ taglib prefix="asset" uri="futuretense_cs/asset.tld"%>
<%@ taglib prefix="ics" uri="futuretense_cs/ics.tld"%>
<%@ taglib prefix="render" uri="futuretense_cs/render.tld"%>
<%@ taglib prefix="user" uri="futuretense_cs/user.tld"%>

<cs:ftcs>

<%-- Record dependencies for the SiteEntry and the CSElement --%>
<ics:if condition='<%=ics.GetVar("seid")!=null%>'>
<ics:then>
<render:logdep cid='<%=ics.GetVar("seid")%>' c="SiteEntry"/>
</ics:then>
</ics:if>

<ics:if condition='<%=ics.GetVar("eid")!=null%>'>
<ics:then>
<render:logdep cid='<%=ics.GetVar("eid")%>' c="CSElement"/>
</ics:then>
</ics:if>

<%-- log in as firstsite--%>
<user:login username="firstsite" password="firstsite"/>
<%-- create the Dimension asset (this can be done manually) --%>
<asset:create name="en_US" type="Dimension"/>
<asset:setsubtype name="en_US" value="Locale"/>
<asset:set name="en_US" field="name" value='en_US'/>
<asset:set name="en_US" field="description" value='US English'/>
<%-- enter your site's pubid below --%>
<ics:setvar name="primarypubid" value="1112198287026"/>
<asset:save name="en_US"/>

<%-- look up the id of the Dimension asset you just created --%>
<asset:get name="en_US" field="id" output="en_US.id"/>

<%-- get a list of all Content_C assets in the site, and assign a dimension to each
of them --%>
<asset:list type="Content_C" list="allContentAssets" pubid="1112198287026"/>
<ics:listloop listname="allContentAssets">
<ics:listget listname="allContentAssets" fieldname="id" output="id"/>
<asset:load type="Content_C" objectid='<%=ics.GetVar("id")%>' name="tempName"
editable="true"/>
<asset:adddimension name="tempName" dimensionid='<%=ics.GetVar("en_US.id")%>'/>
<asset:save name="tempName"/>
</ics:listloop>
</cs:ftcs>
```

25.5 Tips for Using WebCenter Sites Translation Mechanism

Here are some tips that you can use to address some common translation scenarios on your site.

The common two basic scenarios are:

- Editorial teams create and publish content in a primary language first, and then translate and publish translations/localizations as needed in an ad-hoc manner.

- Editorial teams create content and translate it together as a package. When completed, they publish the package all at once.

25.5.1 What Do Customers Want?

- Ability to create content in a master language, wire it up to other related content (which may or may not be in the master language), then either publish it and translate it later, or

- Translate the content and other related bits and pieces, and then publish the whole as a single package.

- Flexibility to apply country-specific rules to the rendered content.

25.5.2 Use of WebCenter Sites Translation Mechanism to Effectively Meet Customers' Requirements

WebCenter Sites has a built-in translation mechanism which uses an assettype called DimensionSets. As a practice, all translated versions of the same content belong to the same DimensionSet instance. This feature also enables designing a fallback tree of locales. When the current locale does not have translated content, another translation which may be available is displayed as the fallback version. A single DimensionSet can contain each locale only once. Using the same locales in a different sequence for different countries is not allowed.

Creating Translations Using Multiple Dimensionsets

To remedy this situation, developers can create different DimensionSets for each country so that they can define an independent fallback logic for each of them. For example, customers would not want to show any English contents on a site like that of France. Whereas a customer based out of The Netherlands would like to include English contents on his site in addition to those in Dutch. In other words, different countries require different fallback mechanisms.

Hence, if a site is targeted toward 20 countries, then there would be 20 DimensionSets (with perhaps 30 or 40 or more locales in total). To support multiple DimensionSets, the rendering template code should be able to determine which DimensionSet to use. The translate tags require the DimensionSet name for the current translation. The following two properties make it possible to use multiple DimensionSet:

- The name of the DimensionSet is the value of the country code. For example, `uk` for the United Kingdom, `de` for Deutchland, and so on.

- All web-referenceable assets have a URL path that explicitly includes the country code (for example, `www.mysite.com/uk/helloworld`, and `www.mysite.com/de/helloworld`, or alternatively one can support different subdomains such as `uk.mysite.com/helloworld`, and so on.

Note:

It is generally considered a bad practice to render different content for the same URL for different users based on their accept-language value or IP geo-location. The reason is that SEO rankings are likely to drop as search engines such as Google may consider such a strategy as a form of bait-and-switch.

With the above two features in place, developers just parse the URL request to extract the country value then pass the country value as an additional argument which can then be used by the Template to specify which DimensionSet to use for the current asset. (It is common for some customers to choose a primary site that has no country code which we can assume is the default.)

Another option is that our code honors the preferred locale cookie which the visitor may have set on their browser.

When Customers Want the Same Content on Websites on Which Other Languages are Used

Some customers want virtual URLs to be supported wherein an asset can be published as .../us/helloworld. These customers also want that the same content is rendered (without needing to translate it) on other English-speaking country sites, for example, .../uk/helloworld, .../ca/helloworld, .../au/helloworld, and so on. The latter requirement can be handled by creating a global (or master) version of an asset. So instead of including only ISO locales in our DimensionSet hierarchy, developers can include made up locales as required. The following CH (Switzerland) DimensionSet: example explains the use of global DimensionSets:

```
global
en_GLOBAL
en_CH
de_GLOBAL
de_CH
fr_GLOBAL
fr_CH
```

The above hierarchical DimensionSet can work as follows:

- When editors require locale-specific differences they can make ad-hoc localized translations as needed. For example, en_CH, de_CH, or fr_CH.

- When language-specific differences need to be available to other DimensionSets, developers can make ad-hoc global translations instead. For example, en_GLOBAL, global-de, fr_GLOBAL, and so on.

A fixed hierarchy like the above does not provide the ability to have exceptions to the fallback logic.

The above example can be extended to also include a regional fallback, for instance, Latin American Spanish es_LAD and South Asian English en_AS. This allows marketing to create content that is localized for a Latin America audience and syndicate this content to many country sites without displaying it on other international sites as it would if en_GLOBAL were used.

Enabling Search for Locale-Specific Content

How do query-based searches compile and render lists of assets on a per-locale basis while taking into account all the various fallback logics? For example, if a visitor is on

a French webpage, probably they only want to see contents that are either `fr_FR` or `fr_GLOBAL`. Duplicate translations in the same DimensionSet should be avoided.

Solution

For queries to return dynamic lists of assets based on searchable attributes, the assets must have attributes that can be used to constrain the search to a specific locale. To enable this, we must convert each asset's selected locale (which is not an attribute as such) into a multi-valued attribute named `SearchableLocale` via a custom flex filter. For instance, an asset whose locale is `fr_FR`, store a single value of `SearchableLocale=fr_FR` since for most clients `fr_FR` would be at the leaf node of the fallback hierarchy for France. However, an asset whose locale is `fr_GLOBAL` should store `fr_FR` as well as any other French-speaking locales across all French-speaking countries. In this way, this global French asset can show up on multiple French language sites via any constrained queries. As an example, for a European-only website, a `fr_GLOBAL` asset would upon save create both `fr_FR` and `fr_CH` `SearchableLocale` values via its flex filter and thus show up on the France (FR) and Switzerland (CH) sites.

> **Note:**
>
> The above denormalization of fallback values into an asset attribute allows you to perform fewer queries at runtime. When marketing requires changing the fallback logic, all data values need to be recomputed. Proceed with caution when designing your solution.

Hiding a Global Asset on Certain Countries Sites

Many customers desire the flexibility to hide a global asset from certain countries. Thus a `HideFromTheseCountries` attribute should be added to the definition for each translatable asset. A custom attribute editor should also be created to restrict the list of countries shown in the editorial interface on which the current asset can be hidden based on the locale selected. That is, if the current locale were `fr_FR`, then the list of countries to hide would be null. Whereas if the locale chosen were `fr_GLOBAL`, the list of countries might include `FR`, `CH`, and so on -- depending on the number of DimensionSets created to support the all the country sites.

With the combination of the custom `HideFromTheseCountries` attribute/attribute editor and a custom flex filter that denormalizes the optimized locales as multi-valued `SearchableLocale` attribute, any queries in rendering Templates only need to add the current locale as a constraint. For example, on a webpage whose locale is `fr_FR`, you might want to show the Latest News in a right-rail pagelet. This pagelet will query the News assets where `locale=fr_FR` and sort the list by date. There might be duplicates since the `fr_GLOBAL` asset is also translated into `fr_FR` and thus both are returned in the list. To remove duplicates, use the translate tag `<dimensionset:filter>` which takes an `iList` from the constrained query that filters out duplicates.

Handling Caching Issues

If you publish all translations at once, any change to any existing translation will cause those pagelets dependent on that asset to expire at publish time. If you publish a global version of a language, and then later create localized versions of that asset, the system will not know to update things since no dependencies were recorded at the last rendering of the pagelet.

A simple example is, let us say a list of the latest press releases on a French page, and let's also assume that all of the assets being rendered are `fr_GLOBAL`. We can even further assume that the links are explicitly defined as named associations, thus all the dependencies are known at the time of rendering/caching. When someone translates one of those assets and publishes it, the desired behavior is that the link would now be to the localized translation, not to the global version. But because there is no way to log a dependency to an asset that did not exist when that pagelet was last rendered, there is no way for the system to know to update that one link. And thus nothing uncaches when you publish the new translation.

The simple solution is to update the master asset whenever a new child is added to a DimensionSet. To do this, one must implement a custom publish listener that performs these tasks in the background whenever a new asset is published that might affect an existing DimensionSet. There is no need to update the master asset when a new translation is created because it might take days or weeks before the asset gets approved and published.

Also be aware that any automatic approval mechanism must also deal with the situation where the master asset may be checked out or not yet approved.

Part IV

Developing Mobile Websites

Mobility enables content contributors to create, preview, and deliver websites to a variety of mobile devices such as phones and tablets. You, as a developer, design the infrastructure that content contributors will use for creating mobile websites.

Topic:

- Configuring WebCenter Sites to Support Mobile Websites

26

Configuring WebCenter Sites to Support Mobile Websites

Mobility enables its users to create, preview, and deliver websites to a variety of mobile devices such as phones and tablets. You need to configure the Mobility framework to enable WebCenter Sites to support mobile-optimized websites.

Topics:

- Prerequisites for Mobility Developers

- Understanding Key Mobility Concepts

- Prerequisites for Configuring Mobility Features

- Configuring Mobility Features

- Mirror Publishing the Device Repository to Delivery System

- Creating Templates

- Optimizing Images for Mobile Websites

- How Device Detection Works

26.1 Prerequisites for Mobility Developers

You should have an experience with core WebCenter Sites features including site navigations and templates to be able to configure the Mobility framework.

You should also have an understanding of mobile devices and their user agents.

26.2 Understanding Key Mobility Concepts

WebCenter Sites uses a built-in device detection mechanism to identify the device that requests website content. Once the device is identified, WebCenter Sites finds the appropriate site navigation (called site navigation in the previous release) and invokes the correct template to render the website.

The mechanics of this device detection process involve new features, such as device repository, device groups and suffixes, device assets, template variants, and site navigations (which have been extended to support mobile websites). This section describes the concepts behind the new features. Later sections provide procedures for configuring the features to support mobile websites.

Once the features are configured, it is possible in the Oracle WebCenter Sites: Contributor interface to create, preview, and deliver mobile sites, such as the site shown in the figure below. In this figure, the Home page of the avisports sample site is displayed in the context of three devices (multi-device preview).

Figure 26-1   Preview of avisports Site Home Page in the Context of Multiple Devices in the Contributor Interface
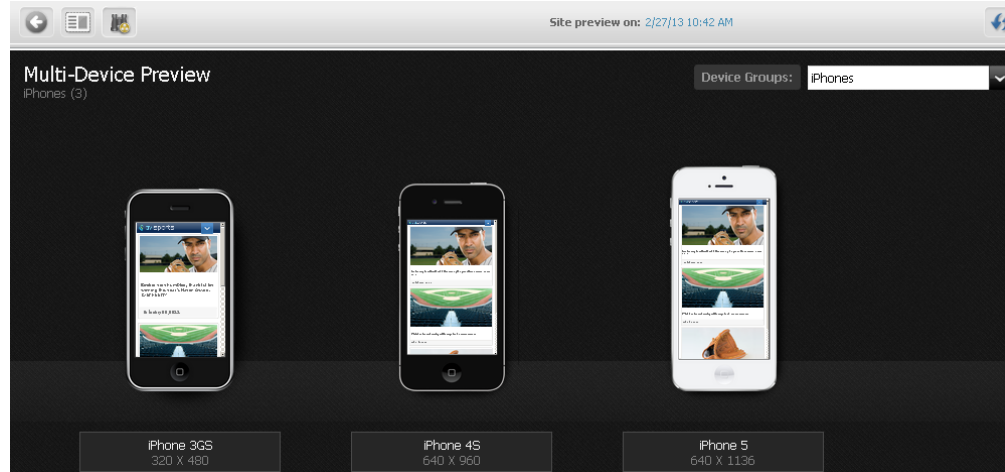
Topics:

- About Device Repository

- About Device Groups and Suffixes

- About Device Assets

- About Site Navigations

- About Mobile Templates

26.2.1 About Device Repository

The device repository is a file that WebCenter Sites uses to detect mobile devices. The device repository contains the properties of devices and uniquely identifies each device based on its user agent. WebCenter Sites detects devices by matching the user agent of the device to the user agent that is specified in the device repository.

Note:

A user agent is a software agent that acts on behalf of users. The format of a user-agent string is a list of product tokens (keywords) with optional comments. Most browsers specify the following format:

```
Mozilla[version] (system and browser information]) [ platform] ([platform
details]) [extensions]
```

For example:

```
Mozilla/so(iPad;u;CPUOS 3.2.1 like Mac OSx;en-us)ppleWebkit/
531.21.10(KHTML,like Gecko) Mobile/7B405
```

The two types of device repositories are:

- `devices.xml`: This is the default repository included with WebCenter Sites. This repository is updated at the time of a product release and includes only popular devices. You can add more data to this repository as and when required.

- WURFL: This is a third-party device information database from ScientiaMobile. WURFL is much more comprehensive than `devices.xml`, and it is updated regularly by ScientiaMobile. We recommend using WURFL to ensure you have the latest devices. A licensed copy can be obtained from ScientiaMobile. It is not included with WebCenter Sites.

For procedures about using the device repository, see How to Configure the Device Repository.

26.2.2 About Device Groups and Suffixes

Device Groups enable features-based grouping of devices. A device group is an asset that defines a collection of devices with common characteristics, for which a common website can be delivered. For instance, all Touch phones could belong in one group, whereas NonTouch (QWERTY) phones could be in a separate group, as illustrated in the next figure. So, one website is delivered to all the Touch devices. Another website is delivered to all the NonTouch (QWERTY) devices. Therefore, two sets of templates must be coded: one set for the group of Touch devices, and another set for the group of NonTouch devices.

Figure 26-2   Device Type-Based Grouping
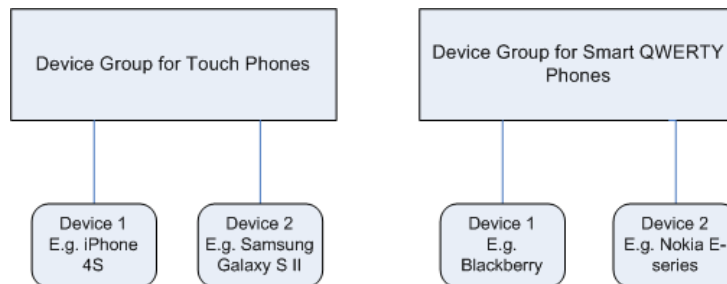
To support template variants, the concept of suffixes has been introduced. Suffixes are used to associate templates to the correct device groups. The association is made when the template and device group have the same suffix, such as `_Touch` (or `_NonTouch`). For example, if the base template `HomeLayout` is used to render the desktop website, you would create template variants, that is, templates named `HomeLayout_suffix`. In this example, you would create the `HomeLayout_Touch` template to render content on devices in the Touch device group. You would also create the `HomeLayout_NonTouch` template to render content on devices in the NonTouch device group.

At runtime, WebCenter Sites will match the suffixes of the device groups to the names of the template variants. When `HomeLayout` is requested from a Touch device, WebCenter Sites will use the `HomeLayout_Touch` template to render the website. This is part of device detection.

> **Note:**
>
> A template without a suffix is called the *base template*. Templates with suffixes are called *variants of the base template*. The base template must exist before its variants can be created.

Suffixes are also used to associate device groups to site navigations, which enables you to implement different website navigation for devices in different device groups.

Templates are coded with the `device:siteplan` tag to specify website navigation on the delivery system.

To create a device group, you will create an asset of type `DeviceGroup`. In the process, you will specify:

- A suffix.

- Either the registered device name(s), or criteria that WebCenter Sites will use to associate devices to the group you are creating.

> **Note:**
>
> Multiple device groups should be prioritized for proper matching of devices to device groups during device detection.

See How to Configure Device Groupsand How to Prioritize Device Groups

26.2.3 About Device Assets

Any mobile device can be represented by a device asset, which enables previewing of mobile website content in the context of the mobile device, in the Contributor interface. Device assets are used strictly to support preview. Even when devices are similar enough to be gathered in the same device group, they display variations (for example, screen size). Previewing in the context of a device asset enables content contributors to ensure that content will be rendered properly on the corresponding real device.

A device asset consists of an image and a user agent. The user agent is used to associate the device asset to (1) a real device in the device repository and (2) to a device group with matching criteria. For an example of associations illustrating the relationship between device groups and device assets, see this figure:

Figure 26-3   Devices Associated with Device Groups
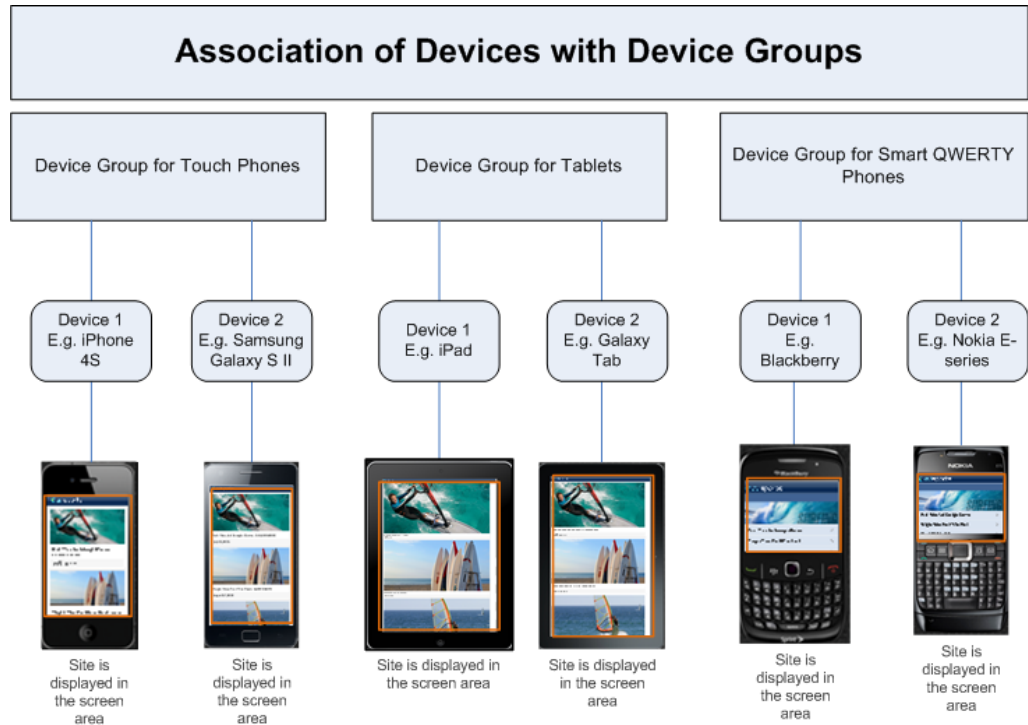
Device assets are associated to device groups by device detection. A device asset that matches the criteria specified in two or more device groups is automatically associated to your preferred (highest-priority) device group. A device that fails to match criteria in all device groups is automatically assigned to the Default device group (used for serving websites to desktop browsers). Once a device is associated to a device group, templates associated with that device group can render website content in preview mode. The content is superimposed on the device image.

Note:

Ensure that content contributors are aware of the following preview behavior:

- While preview of web pages in mobile devices works in all browsers, some features are not displayed correctly if there is a mismatch between the browser and the user agent. For example, if the Contributor interface is opened in Internet Explorer, whereas the user agent is for FireFox on Android, the browser will use the Internet Explorer engine to render HTML. Therefore, previews in the Contributor interface are likely to differ from the look of the real pages on the real mobile device.

- Preview renders pages as they would be displayed in full-screen mode on real devices.

See How to Create Device Assets.

26.2.4 About Site Navigations

A site navigation defines the navigational hierarchy of a website. For example, the next figure shows **Default**, **Touch**, and **NonTouch** site navigations for the avisports sample site in the Contributor interface.

Figure 26-4   Site Navigations in the Contributor Interface
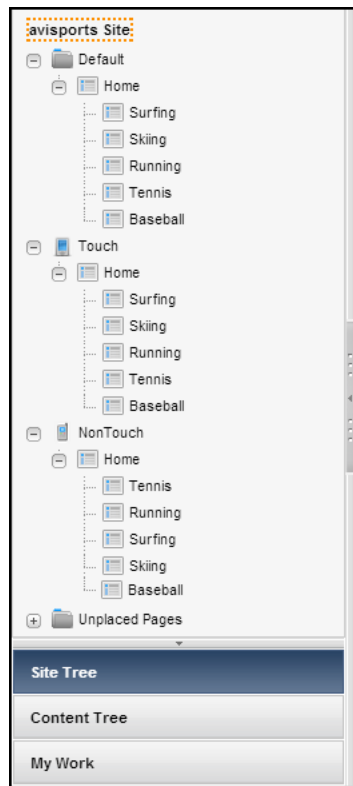
When creating a site navigation, you associate device groups to that site navigation by selecting a shared suffix. Site navigation can then be served to devices in those device groups. (Once selected, the suffix is no longer available for other site navigations.)

You have two basic approaches to creating site navigations:

- Create a single site navigation for all devices across all device groups.

- Create different site navigations for different device groups. The content can be reused across site navigations, or it can be different.

See How to Create Site Navigations.

26.2.5 About Mobile Templates

You have two basic approaches for creating templates that render mobile websites:

- Create a single set of templates that adapt to all mobile devices (and therefore all device groups). Adaptive templates adapt to the screen resolution of a device and render content suitably. In this scenario, there is no need for creating templates with suffixes.

- Create different templates for different device groups. (An example was described in About Device Groups and Suffixes.) This scenario requires you to create two sets of templates: One set contains the base templates (without any suffixes). The other set contains the template variants (templates with suffixes). To create a template variant, you append `_suffix` to the name of the base template.

  For example, the template name for a device group called iPhones, whose suffix is `Touch`, will be `BaseTemplateName_Touch`. This naming convention and the matching of suffixes for device groups and templates ensures that WebCenter

Sites routes requests from mobile devices to the correct template variants. The base template is used whenever a mobile template variant does not exist for a specific device group.

> **Note:**
>
> WebCenter Sites does not recognize templates with two suffixes (such as _Touch_NonTouch). They are not considered variants of existing templates. Such templates are not available through the **Choose Page Layout** option on the asset toolbar in the Contributor interface, and therefore, they are not used for rendering assets. The base template is used in place of a template that has two suffixes.

You can create templates at any time once you have created the device groups and you know the suffixes. See Creating Templates.

26.3 Prerequisites for Configuring Mobility Features

To be able to configure Mobility features, you should have the necessary credentials and information. For example, the GeneralAdmin role, device repository, and so on.

- The only user who is automatically granted access to the **Mobility** tab in the Admin interface is the user with the GeneralAdmin role.

  > **Note:**
  >
  > The MobileSitesDeveloper role is no longer available. If you have upgraded from 11.1.1.8.0 release, be aware that MobileSitesDeveloper role acts as a user-defined role and has no particular significance. Therefore, you must grant access to the **Mobility** tab to the GeneralAdmin role.

- An understanding of which device repository you will be using. See How to Configure the Device Repository.

- Information about the device groups you will be configuring.

  - The names of device groups in which to organize different devices.

  - Which capabilities (such as user agent, touch, tablet, and screen resolution) you will use to identify and group devices. For more information as to which data you will provide, see How to Configure Device Groups.

  - Which suffix you will use for each device group. A suffix is required for each device group. (You will append the same suffix to the names of the template variants for these device groups. You will also select the same suffix for the site navigation.)

  - Whether you will use custom filters to apply conditions on capabilities that are not listed by default. See How to Create Custom Filters for Device Group Criteria.

- The list of devices your contributors will be using to preview the mobile website. WebCenter Sites comes with several device assets that represent many commonly used devices.

To create your own device assets, first create your own images of the real devices, and look up the user agents of the real devices. You will use these images and user-agent information to create device assets. Also create their thumbnail images for use in the device selector panel in the Contributor interface.

- Effective site navigations for the mobile versions of your website.

26.4 Configuring Mobility Features

Some of the tasks you perform to create the Mobility infrastructure are activating the device repository, configuring device groups, creating custom filters for device groups.

See these topics:

- How to Configure the Device Repository
- How to Create Custom Filters for Device Group Criteria
- How to Activate Your Device Repository
- How to Configure Device Groups
- How to Prioritize Device Groups
- How to Create Device Assets
- How to Create Site Navigations
- How to Organize Site Navigations

26.4.1 How to Configure the Device Repository

You have the option to use one of the following device repositories: `devices.xml` (default repository) or WURFL.

- To use `devices.xml`, do the following:

  1. Ensure the file contains the device names and user agents you require. Locate the file in the `DeviceRepository` directory and make the necessary changes.

  2. Activate `devices.xml` by uploading to WebCenter Sites.

     a. Open the **General Admin** tree, expand the **Mobility** node, and then double-click **Device Repository**.

     b. In the Device Repository Uploader form, upload `devices.xml`, and then save.

- To support more capabilities and devices than currently registered in the `devices.xml` file, use WURFL as the device repository. Do the following:

  1. Obtain the WURFL repository in one of the following formats:

     – `WURFL.zip`

     – `WURFL.xml` with a WURFL patch file. The patch file is used to override the content of `WURFL.xml`. For more information about patch files, see http://wurfl.sourceforge.net/.

> **Note:**
>
> WURFL is a third-party device repository. You must purchase a license from ScientiaMobile to use this repository. It is not included with WebCenter Sites.

2. To use the `wurfl.xml` file and patch, configure the patch file before you continue to other steps.

3. Activate the WURFL repository by uploading to WebCenter Sites.

26.4.2 How to Create Custom Filters for Device Group Criteria

WebCenter Sites provides a default custom filter implementation (`DefaultCustomFilter.java`), which takes an XML file as input. To create a custom filter of your own, write an implementation of the `CustomDeviceFilter.java` interface. The XML file for a custom filter is uploaded from the Device Group configuration page, as described in How to Configure Device Groups.

This section includes the following:

- Using the Default DefaultCustomFIlter.java Custom Filter Provided with WebCenter Sites

- Creating Your Own DeviceGroupFilter Implementation

> **Note:**
>
> For information about how to use custom filters, see How to Configure Device Groups.

26.4.2.1 Using the Default DefaultCustomFIlter.java Custom Filter Provided with WebCenter Sites

The default implementation class of a custom filter is called `COM.FutureTense.Mobility.Filter.DefaultCustomFilter`. It takes input in XML format. In the default custom filter implementation provided with WebCenter Sites, a filter is passed only if *all* its arguments are passed. Similarly, an entire custom filter is passed when *all* its filters are passed. So, in any scenario, *all* arguments must be passed for the filter criteria to work.

The following example shows a sample filter XML in which device property names are taken from the WURFL repository:

```
<?xml version="1.0" encoding="UTF-8"?>
<devicefilters>
        <filter name="tabletFilter"
classname="COM.FutureTense.Mobility.Filter.DefaultCustomFilter"> //Filter 1
            <argument name="is_tablet" value="true" datatype="boolean"/> //argument
1 of filter 1
            <argument name="pointing_method" value="touchscreen" datatype="string"
operator="equals"/> //argument 2 of filter 1
        </filter>
            <filter name="flashFilter"
classname="COM.FutureTense.Mobility.Filter.DefaultCustomFilter"> //Filter 2
            <argument name=" full_flash_support" value="false"
datatype="boolean"/> //argument 1 of filter 1
```

```
            </filter>
</devicefilters>
```

In the above sample, `tabletFilter` has two arguments. `argument 1` requires that the value of the `is_tablet` property should be `true`. `argument 2` requires that the value of the `pointing_method` property should be `touchscreen`. `flashFilter` has only one argument which is, the value of the `full_flash_support` property should be `false`. Each argument is a rule and a complete filter. Only when *every* argument is met, a device can match to the device group containing the above custom filter.

This sample XML uses device property names from the WURFL repository. The default device repository (`devices.xml`) XML looks something like the following example (notice that the property names have changed). In this filter XML, there are two filters: `tabletFilter` and `flashFilter`.

```
<?xml version="1.0" encoding="UTF-8"?>
<devicefilters>
      <filter name="tabletFilter"
classname="COM.FutureTense.Mobility.Filter.DefaultCustomFilter"> //Filter 1
           <argument name="tablet" value="true" datatype="boolean"/> //argument 1 of
filter 1
           <argument name="touch" value="true" datatype="string"
operator="equals"/> //argument 2 of filter 1
      </filter>
      <filter name="flashFilter"
classname="COM.FutureTense.Mobility.Filter.DefaultCustomFilter"> //Filter 2
           <argument name="flash" value="false" datatype="boolean" /> //argument 1
of filter 1
      </filter></devicefilters>
```

While creating a custom filter based on `DefaultCustomFIlter.java`, consider the following:

- In a custom filter, possible values of the `datatype` argument attribute are `string`, `number`, `boolean`. Default value is `string`.

    - When `datatype = number`, the possible values of the `operator` attribute are: `<>`, `notequals`, `<`, `lt`, `>`, `gt`, `=`, `equals`. Default value is `equals`.

    - When `datatype = boolean`, the possible values of the `operator` attribute are: `=`, `equals`. Any other value is treated as the reverse of `equals`. Default value is `equals`.

    - When `datatype = string`, the possible values of the `operator` attribute are `=`, `equals`, `%`, `like`, `!=`, `notequals`, `!%`, `notlike`. Default value is `equals`. The following snippet shows the use of the `like` or `%` value for the `operator` attribute:

        ```
        <argument name="pointing_method" value="touch" datatype="string"
        operator="like"/>
        ```

        Here, the value of the `pointing_method` property must contain the word `touch` either as a substring or an entire word.

- The value of the property attributes must be as per the property names in the current device repository (either `devices.xml` or `WURFL.xml`).

26.4.2.2 Creating Your Own DeviceGroupFilter Implementation

The following example shows a Java class that implements the `DeviceGroupFilter` interface and uses the `matches` method. Your custom filter can consist of one or more filters. Each filter can contain zero or more arguments. The custom filter implementations can use the `OR` rule, or any custom logic.

```
public class UserDefinedCustomFilter implements DeviceGroupFilter
{
public boolean matches(DeviceContext context)
    {
        // Logic that returns true/false depending on whether criteria matched or not.
    }
}
```

26.4.3 How to Activate Your Device Repository

To activate your device repository:

1. In the Admin interface, open the **General Admin** tree, then expand the **Mobility** node, and then double-click **Device Repository**.

2. In the Device Repository Uploader form, upload either the default repository (`devices.xml`) or the WURFL repository (`WURFL.zip`, or `wurfl.xml` and its current patch file).

    > **Note:**
    >
    > When you switch the device repository, you are reminded to update the device names that you changed in `devices.xml`, across all device groups.
    >
    > The `WURFL.patch` file is used with the `WURFL.xml` file, regardless of the record for `WURFL.patch` in the device repository table which is still set to `Active=F` (false).

Figure 26-5   Device Repository Uploader Form

3. Click the **Save** icon.

26.4.4 How to Configure Device Groups

To configure a device group:

1. Under the **Mobility** node, expand the **Device Groups** node.

2. Click **Add Device Group**, as shown in this figure:

Figure 26-6   Add Device Group
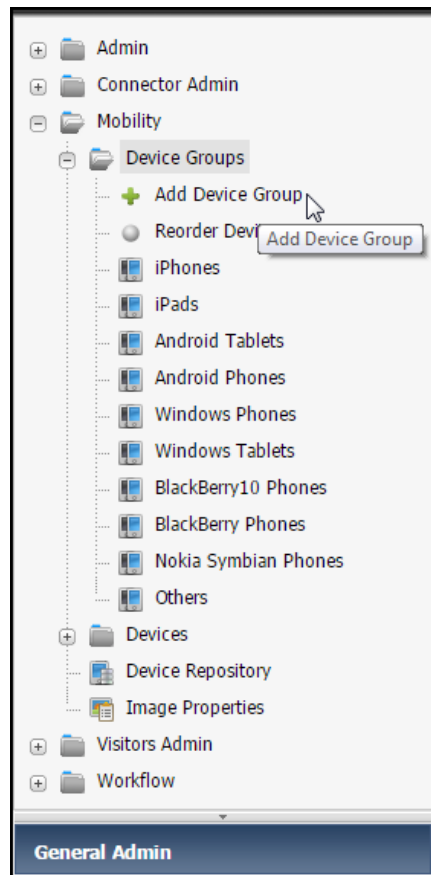
The Device Group form opens.

3. In the Device Group form, select the **Content** tab and do the following:

    a. In the **Name** field, enter a meaningful name for the device group you are creating.

    b. In the **Suffix** section, enter the suffix you plan to use for the templates you will create for this device group. Or, choose an existing suffix if there is any.

        Note:

        This suffix is not editable. To change it later, you will have to recreate this device group and change this suffix.

        ---

    c. In the **Active** field, enable this device group for device detection by selecting **Yes**.

        Note:

        Device groups are global. Once enabled, they become available for all sites. Similarly once disabled, they are disabled for all sites and no longer used in device detection.

        ---

4. Select the **Criteria** tab (see the next figure) to create a set of rules for matching real devices so they can be associated with their correct template variants.

Either provide one or more device names (the rest of the form will be disabled (see the figure below), or omit the device name(s) and fill in the rest of the form.

Figure 26-7   Add Device Names

a. If you choose to enter device name(s), enter the same name(s) that are registered in the device repository. If you uploaded `devices.xml`, enter the device entry name. If you uploaded WURFL, enter the *model_name* of the device.

The **Device Names** field provides *Typeahead* feature so you can choose a registered device name from the list of available devices. As you start typing a device model number, this features shows all the registered devices beginning with the common letters. You can easily scroll and choose from the first 50 devices. To see more devices, please refine your search further.

A device can belong to only one device group at a given time.

b. If you choose to omit device names, fill in the rest of the form, as follows:

i. **User-Agent Section**: This section is disabled if you specify the device names.

In this field, enter just a list of device names, or a combination of user agent, screen dimensions, capabilities, and custom filters. A combination of user-agent `regex`, capabilities, screen dimensions, and custom filter requires a device to meet all the rules to match with the device group.

Enter the exact user agent of the browser from which the incoming request will be sent. Or, enter a regular expression or a substring to match the set of the user agents. For example, to match all iPhone user agents, specify the user-agent `reg exp` as `(m|M)ozilla/5.0(| )\` `(i(p|P)(hone|od|rod).*` This expression will match any user-agent string which contains the `iPhone` Java `regex`. Or, use the substring `iPhone,` which matches all iPhones.

Note:

To see how many devices for the user agent you entered are available in your device repository, click the **Refresh** icon in the **Matching Devices** section.

---

ii. **Capabilities Section**: This section is disabled if you specify the device names. The capabilities are: **Touch Screen**, **JavaScript**, **Dual Orientation**, and **Is Tablet**. Each capability gives you the following options: **Yes**, **No**, **Don't evaluate**.

For example, to match this device group to only Tablet devices that do not support JavaScript, select **Yes** for the **IsTablet** capability; for the JavaScript capability, select **No**; and for the rest of the capabilities, select **Don't evaluate**.

iii. **Screen Resolution Section**: This section is disabled if you specify the device names. Enter the minimum and maximum width and height for the display area (units are in pixels). For example, a maximum width of 640 will match this device group to all devices whose screen resolution width is 640 or less.

iv. In the **Custom Device Filters** section, click **Browse** to choose a custom filter that you might have created to apply conditions on capabilities that are not listed on the **Device Group Criteria** tab.

5. Click the **Save** icon to save your device group.

Your new device group is listed on the **Mobility** tab, at the bottom of the **Device Groups** node (as shown in the next figure).

Figure 26-8   New Device Group
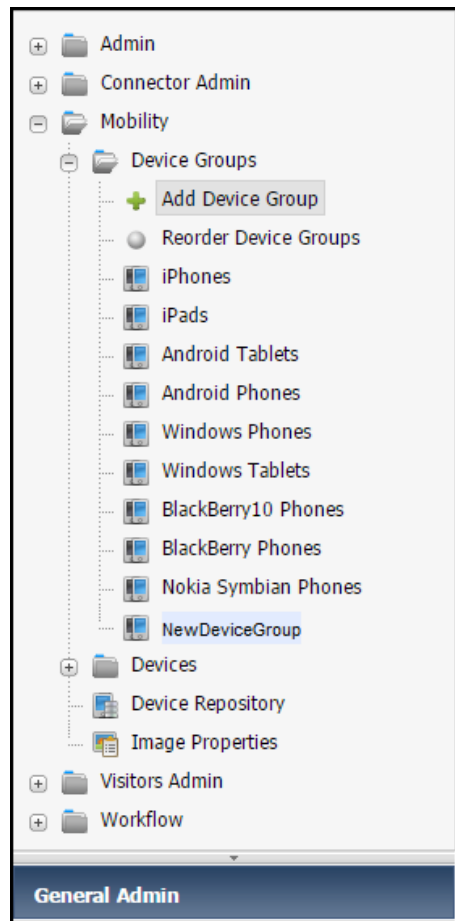
26.4.5 How to Prioritize Device Groups

You must prioritize multiple device groups to enable device detection which automatically associates a real device to the highest-priority device group at runtime.

To prioritize device groups:

1. Under the **Mobility** node, expand the **Device Groups** node.

2. Double-click **Reorder Device Groups** .

Figure 26-9   Reorder Device Groups
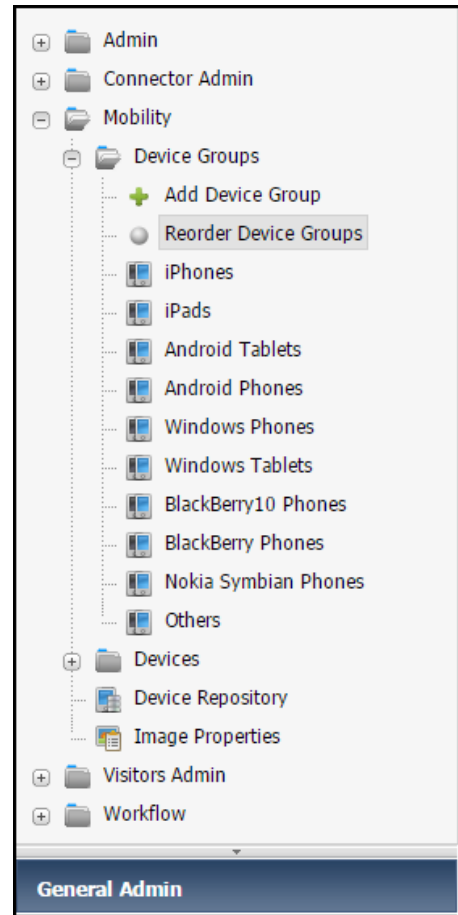
3. On the Reorder Device Groups page, drag and drop the device groups in the order of your preferred priority.

Figure 26-10   Drag and Drop Device Groups
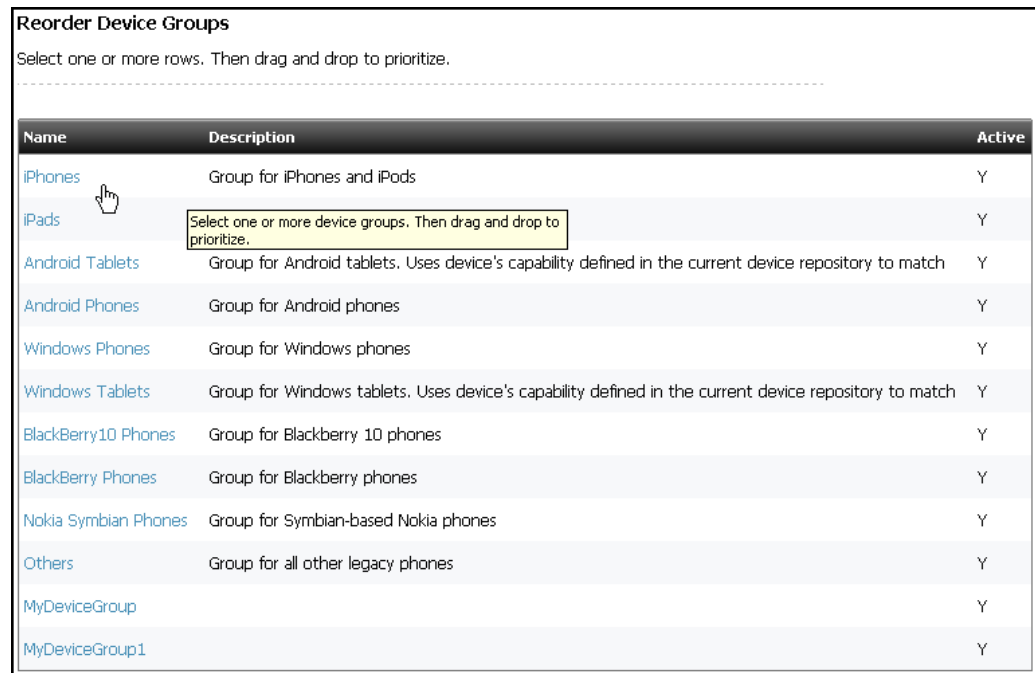
When you have reordered the device groups, the Reorder Device Groups page displays them in the new sequence (such as the one in the figure below).

Figure 26-11   Device Groups Reordered
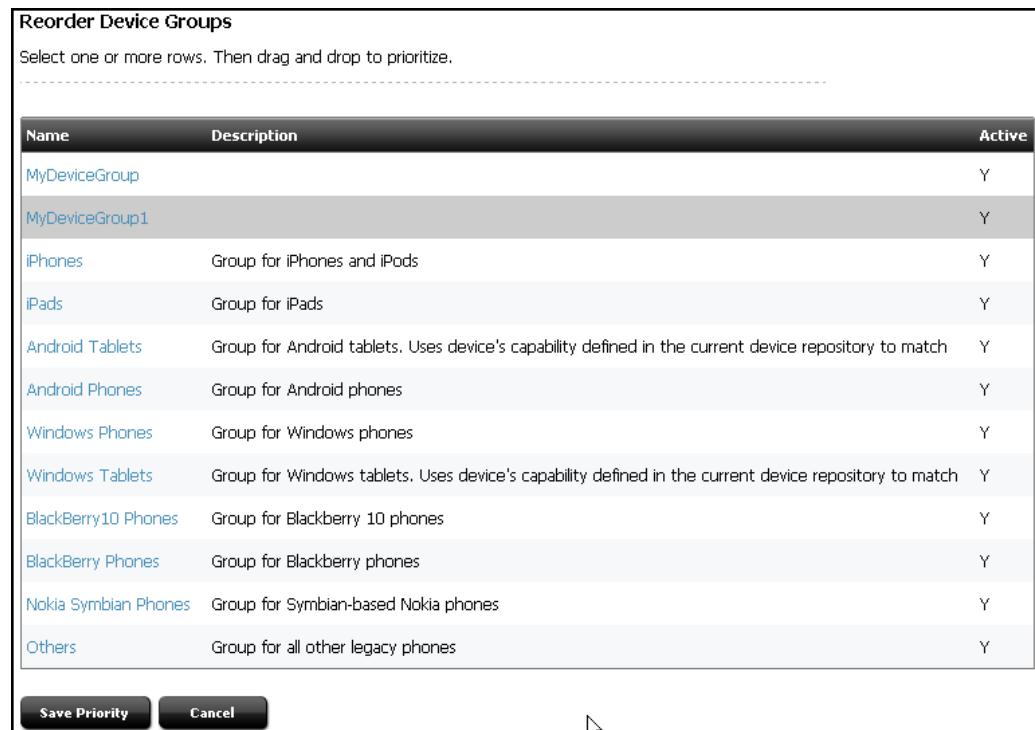
4. Click **Save Priority**.

    The **Device Groups have been re-prioritized successfully** message is displayed.

Once you have prioritized your device groups and created the device assets, verify that device groups and assets are correctly associated by device detection. Open the device group and select the **Devices** tab to view the list of device assets associated with the device group.

26.4.6 How to Create Device Assets

To create a device asset:

1. Under the **Mobility** node, expand the **Devices** node.

2. Click **Add Device** .

Figure 26-12   Add Device
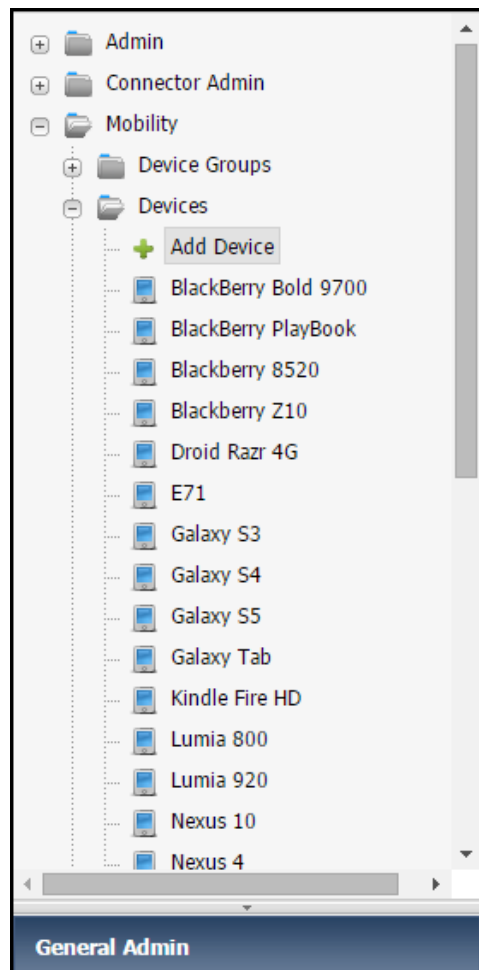
The Device form opens.

Figure 26-13   Device Form - Content Tab

3.   On the **Content** tab:

   a.   In the **Name** field, enter a name for this device asset.

        This name will be displayed in the Contributor interface. It does not have to match the name in the device repository.

   b.   (Optional). In the **Manufacturer** field, enter the name of the device maker company.

   c.   In the **User Agent** field, specify the registered user agent for the device whose image you are adding. You can copy the user agent from the device repository.

   Note:

   The User-Agent field identifies the real device that this device asset represents. This field is used in device detection logic to associate a device with the matching device group of highest priority.

   d.   Click **Test User Agent** to run device detection and confirm that this device matches a particular device group.

   e.   Select the **Enable** option to make this device asset available to device detection mechanism so this asset can be associated to device groups.

   f.   Next to the **Device image** field, click **Browse** to select the device image from the directory in which the image is stored.

   g.   (Optional). Next to the **Thumbnail image** field, upload a thumbnail image to be displayed in the device selector panel that is associated with the preview feature in the Contributor interface. Selecting the thumbnail displays a page preview in the device image.

4. On the **Screen Dimensions** tab, enter the required pixels in the **Height**, **Width**, **Top**, and **Left** fields and pixel ratio in the **Pixel Ratio** field to determine an appropriate dimension of the screen area.

- **Height**: Height of display area within the device image in which you'll preview your site content.

- **Width**: Width of display area within the device image in which you'll preview your site content.

- **Top**: Top margin for display area within the device image in which you'll preview your site content.

- **Left**: Left margin for display area within the device image in which you'll preview your site content.

- **Pixel Ratio**: Ratio between physical pixels and logical pixels. For instance, iPhone 7 has a pixel ratio of 2 because the physical linear resolution is double the logical linear resolution.

As you enter the pixels in each field, the screen area of the device image begins to reset accordingly. This feature lets you determine the exact dimension of the device display area on the spot.

Figure 26-14   Screen Dimensions Tab

5. Click the **Save** icon on the form to create the new device.

The success message is displayed.

26.4.7 How to Create Site Navigations

Whether you need multiple site navigations or a single site navigation depends on your design approach. When creating a site navigation, you will associate device groups to this navigation by selecting a common suffix. Once selected, that suffix is no longer available for other site navigations.

To create a site navigation:

1. In the Admin interface, open the **General Admin** tree, expand the **Admin** node, then the **Sites** node, and then the site for which you are creating the site navigation.

2. Expand the **Site Navigations** node.

3. Double-click **Add New** .

Figure 26-15 Site Navigations Node in the Admin Tab
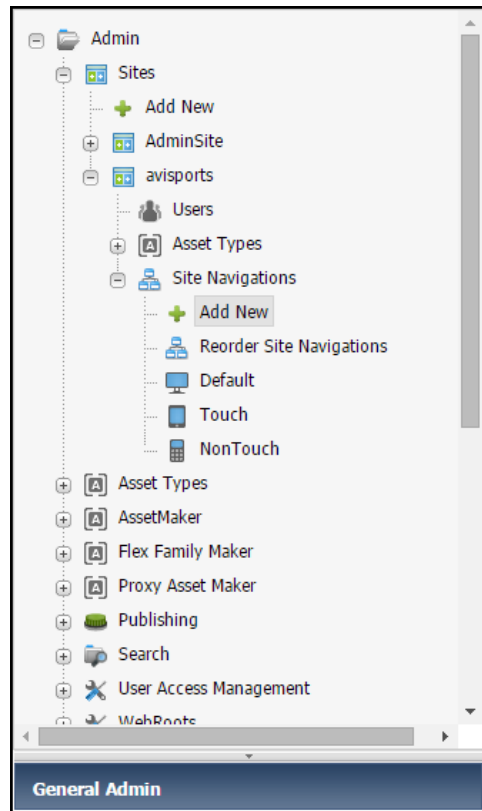
The Add Site Navigation form opens.

4. In the **Name** field, enter a meaningful name for your site navigation.

5. (Optional) In the **Description** field, enter a description for your site navigation.

6. To associate device groups with this site navigation, go to the **Suffix** section, and select a suffix used by those device groups.

Note:

The **Suffix** section lists only suffixes that are not assigned to any site navigations.

The **Associated Device Groups** panel lists the device groups that are associated with your selected suffix.

Figure 26-16   Associated Device Group Panel

7. Add a meaningful icon to your site navigation to help content contributors identify whether the pages opened in Web or Form mode belong to this particular site navigation. This feature is quite useful when working with multiple site navigations. To add an icon, next to the **Icon** field, click **Browse**, then navigate to the directory in which the icon is located. The recommended icon image size is 16 pixels * 16 pixels.

In the Contributor interface, this icon will appear on left of the site navigation and of its pages when they are opened in Web or Form modes. For example, see this figure:

Figure 26-17   Site Navigation Icon
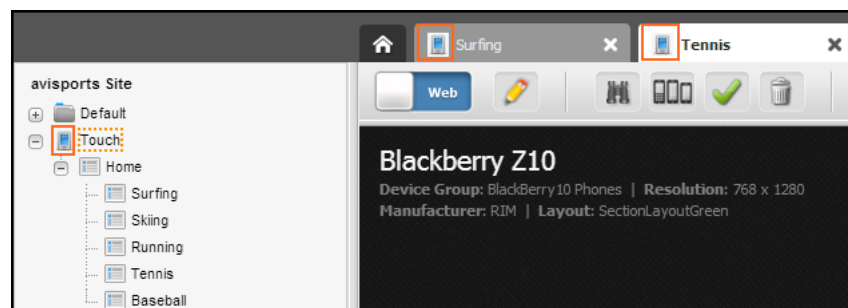
8. Click **Add** to complete the site navigation.

A page similar to this figure opens.

Figure 26-18   New Site Navigation Created
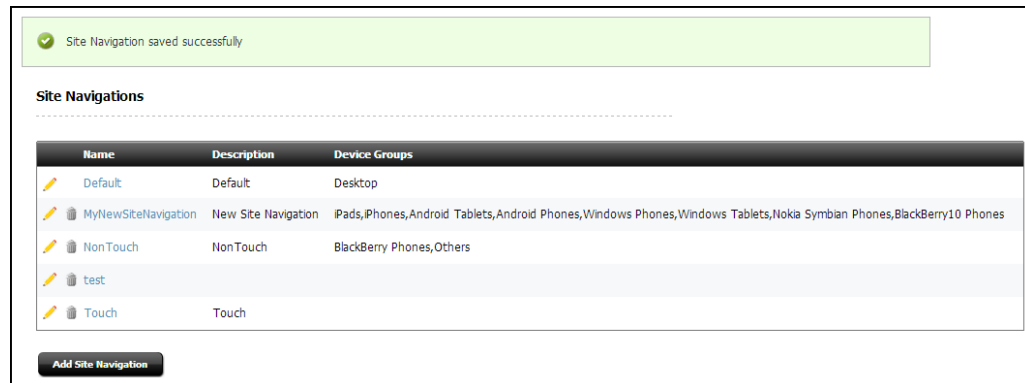
9. Click the **Site Navigation** tab to see the newly created site navigation.

   To modify the site navigation, double-click it under your site's node on the **Admin** tab. On the Modify Site Navigation page, make the changes and then click **Modify**.

26.4.8 How to Organize Site Navigations

Reorder site navigations to display them in a specific order in the Admin and Contributor interfaces.

To organize site navigation:

1. In the Admin interface, under the **General Admin** tab, expand the **Admin** node, and then expand the **Sites** node and the site for which you are prioritizing the site navigations.

   > **Note:**
   >
   > If you are assigned the SiteAdmin role, use the **Site Admin** tab instead.

2. Under **Site Navigations**, double-click **Reorder Site Navigations**.

Figure 26-19   Reorder Site Navigations
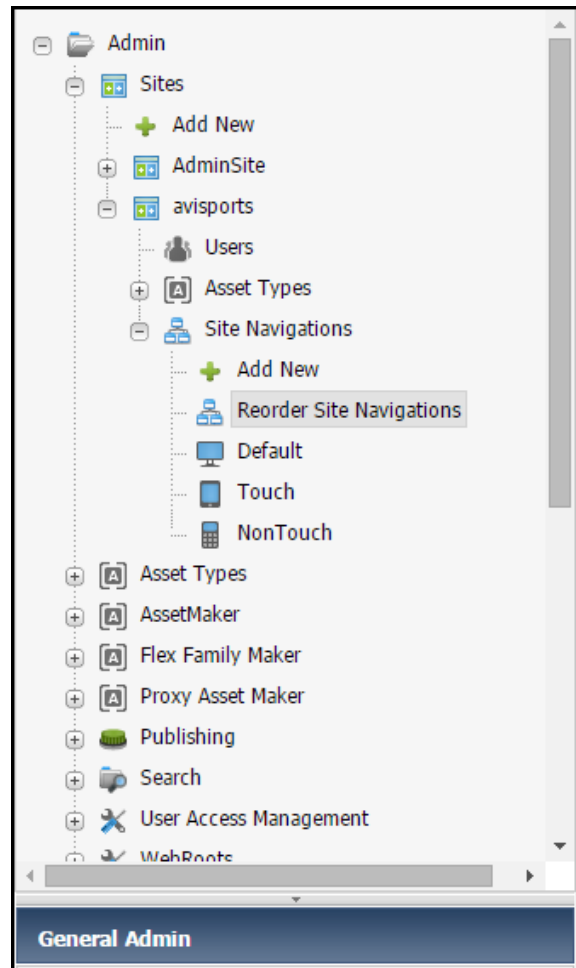
3. On the Reorder Site Navigations page, drag and drop site navigations to order them in the preferred sequence.

Figure 26-20   Drag and Drag Site Navigations

When you have reordered the site navigations, the Reorder Site Navigations page looks something like this figure.

Figure 26-21   Site Navigations Reordered

4. Click **Save**.

The **Modification was successful** message is displayed.

26.5 Mirror Publishing the Device Repository to Delivery System

After you've configured the infrastructure for your mobile site, remember to mirror publish the device repository to the delivery system.

> **Note:**   After modifying and mirroring a device repository, you can trigger the reassociation of devices with device groups on the target system by opening a device group for editing and saving it without necessarily changing any values.

To mirror publish the WURFL device repository:

1. Under the **Mobility** node, double-click the **Device Repository** node.

2. In the Device Uploader screen, under **WURFL**, select the **Single Zip upload** option and upload the `wurfl.zip` file.

3. Click the **Mirror** icon located next to the **Save** icon.

The available destination options are displayed.

The destinations that are up and running have the Green icon, and those that are not running currently, show the Red icon. When a destination's icon is Red, the check box is disabled.
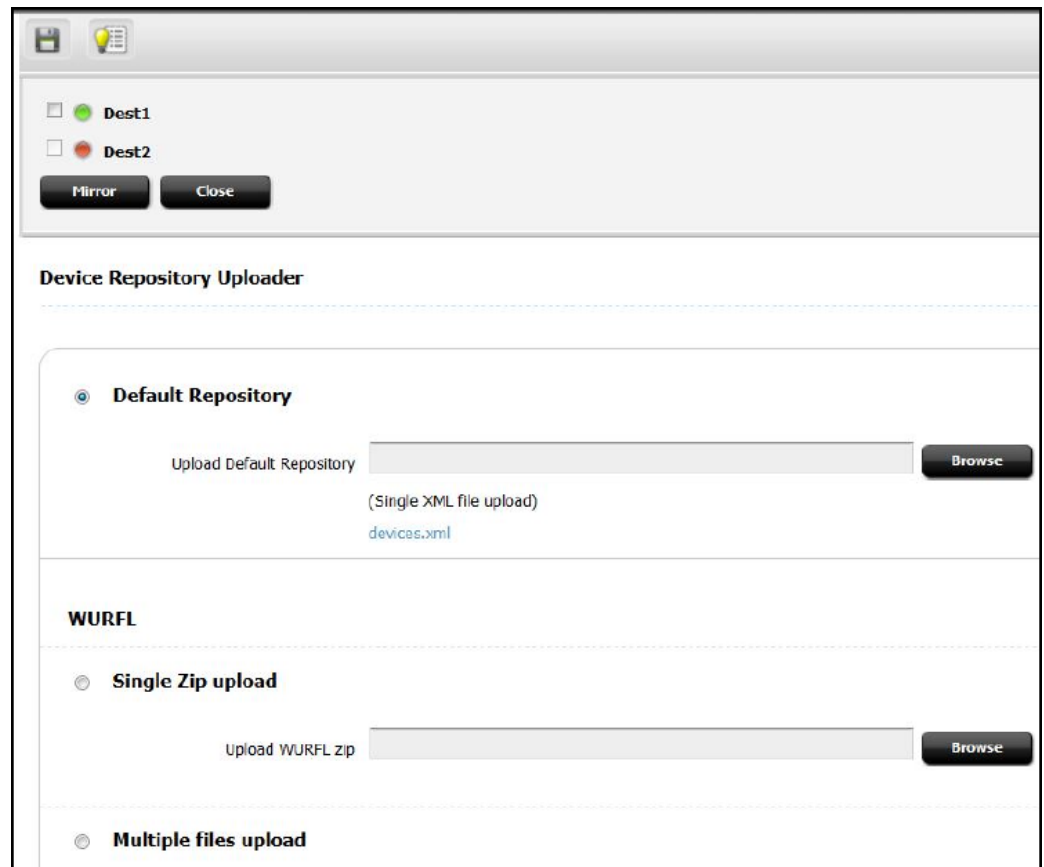
4. Select the check box for the desired destination option.

Note:

Before performing the next step, ensure that WURFL.jar is already available on the destination machine. Mirroring the WURFL device repository without placing a copy of the WURFL.jar on the delivery system prevents the device groups from functioning properly.

---

5. Click **Mirror**.

   The chosen device repository is mirrored to the selected destination. A success message is displayed.

   The selected destination's indicator turns Grey after the repository is mirrored.

26.6 Creating Templates

You can create just one set of templates that works for all mobile device or different sets using the suffix feature of WebCenter Sites.

So, you have two basic approaches for creating templates that render mobile websites:

- Create a single set of templates that adapt to all mobile devices (and therefore all device groups).

- Create different templates for different device groups. This scenario requires the use of suffixes. This section discusses this second approach.

See these topics:

- [Basic Guidelines for Creating Template Variants](#)
- [Understanding Mobility Tags](#)
- [Tags Modified to Support Device Detection and Page Rendering](#)
- [Creating Template Variants](#)

26.6.1 Basic Guidelines for Creating Template Variants

When creating different templates for different device groups, note the following requirements:

1. Create the base template (the template that renders the desktop website).

2. Create the template variant by using the name of the base template and appending _*suffix*.

3. Use the suffix (the d parameter) as one of the cache criteria to cache pages based on your templates.

26.6.2 Understanding Mobility Tags

The table below describes the Mobility tags you will use when creating templates. See the *Oracle Fusion Middleware Tag Reference for Oracle WebCenter Sites Reference*.

Table 26-1   Tags for Mobility

Tag	Description
`<device:load name="<Name of Current Device>" />`	Detects the current device and loads the device information. Similar to `asset:load`.
`<device:get name="<Name of Current Device>" property="useragent|devicegroup|suffix" [output="propName"] />`	For a loaded device, this tag returns the value of one of the following properties: `useragent`, `devicegroup`, or `suffix`. If the optional attribute `output` is provided, this tag sets the value of the property in the `output` variable. If the `output` attribute is absent, this tag sets the value of the property to a variable with the name of that property.
`<device:if name="<Name of Current Device>" property="touch" value="true" [datatype="boolean"] [operator="equals"] > ... conditional code for touch devices only ... </device:if>`	For the currently loaded device, this tag allows you to specify a condition and code that will be executed when the condition is met. The default value for the optional attribute `datatype` is `String`, and the default value for the optional attribute `operator` is `"="`.
`<device:hascapability name="<Name of the Device Loaded Earlier>" capability="<WURFL_CAPABILITY_NAME>" > conditional code </device:hascapabiilty>`	For the currently loaded device, this tag checks if this device supports the capability specified in this tag. The code in the tag is executed if the device supports this capability.

Table 26-1   (Cont.) Tags for Mobility

Tag	Description
`<device:capability name="<Name of the Device Loaded Earlier>" capability="<WURFL_CAPABILITY_NAME>" output="capabilityValue" />`	For the currently loaded device, this tag sets the value of the specified capability in the `ics` scope.
`<device:siteplan output=<NAME_OF_VARIABLE_HOLDING_RESULTING_SITEPLAN_ID> [pubid = < %=ics.GetVar("pubid")%>] [site=< %=ics.GetVar("site")%>] [d= <%=ics.GetVar("d") %>] />`	Use the `device:siteplan` tag to look up the site navigation for any device. The tag takes the `d` parameter, which is a device group suffix. At runtime, WebCenter Sites computes the value of the `d` parameter by using device detection. The tag then uses the value of the `d` parameter (that is, the suffix) to find the site navigation with the same suffix and to return the ID of that site navigation. The site navigation ID is used by other tags to construct website navigation.

26.6.3 Tags Modified to Support Device Detection and Page Rendering

The following tags include Mobility-specific attributes:

*   `render:getpageurl`

*   `render:calltemplate`

*   `render:gettemplateurl`

*   `render:gettemplateurlparameters`

*   `insite:calltemplate`

*   `satellite:page`

*   `satellite:link`

The attributes are:

*   `d`, the suffix of a device group.

*   `resolvetemplatefordevice`, which appends the device group suffix to the template name. The default value is `true`. If the value is set to `false`, the suffix is not appended to the template name and the base template is loaded.

The `d` parameter is automatically passed down the chain in each of the tags above. The tags call the correct template variant by basing their call on the passed `d` attribute.

In the following example for the avisports sample site, `c=Page` and `TestSite` is the current site.

```
<render:calltemplate tname='Detail' args="c,cid,p,d,locale,form-to-render" />
(i) for d=Desktop (default) or blank, the following template is called:
        TestSite/Page/Detail
```

```
(ii) for d=Touch, the following template is called:
          TestSite/Page/Detail_Touch
```

For information about how the d attribute is used, see How Device Detection Works.

26.6.4 Creating Template Variants

You create the template variants by copying, modifying, and adding a suffix to base templates. This suffix feature lets you create both, a single template variant and template variants in bulk for suffixes for which no variants exist.

This section includes the following topics:

- How to Create a Variant of a Single Template

- How to Create Template Variants in Bulk

> **Note:**
>
> Suffix-based search lets you easily locate templates of a certain suffix. That is, click **Search** on the menu bar and then choose **Find Template** from the **Search** table. On the **Search for: Templates** screen, choose **Suffix** from the **Search** drop-down list and the desired suffix from the **for** drop-down list. The search results also include deactivated or deleted templates. (Remove these templates from your site so they do not show up in search results again.)

26.6.4.1 How to Create a Variant of a Single Template

To create a template variant:

1. Determine if a variant of a template already exists. Navigate to the template's **Inspect** mode's **Template Variants** section.

   The **View** link is displayed under the **Action** column if a template variant exists for a particular suffix, otherwise the **Create** link is displayed.

Figure 26-22   Template Variants

2. To create a variant of a template, click the **Create** link.

The Template form is displayed.

Figure 26-23   Template Form

3. Perform Name and Describe the Template Asset of Creating Template, CSElement, and SiteEntry Assets.

26.6.4.2 How to Create Template Variants in Bulk

To create template variants of multiple templates:

1. On the **Dev** tab, expand **Template**, then double-click **Bulk Copy Templates**.

2. On the **Bulk Copy Templates** page, choose the required suffix from the **Suffix** drop-down list.

3. In the **Asset Types** box, select an asset whose templates you wish to copy.

   This displays the associated base templates in the **Available Base Templates (n)** box.

4. Select the templates you wish to copy. Choose one, many, or all templates displayed in the **Available Base Templates** box, as required.

The **Selected Base Templates** box is populated with the chosen templates.

Figure 26-24   Bulk Copy Templates
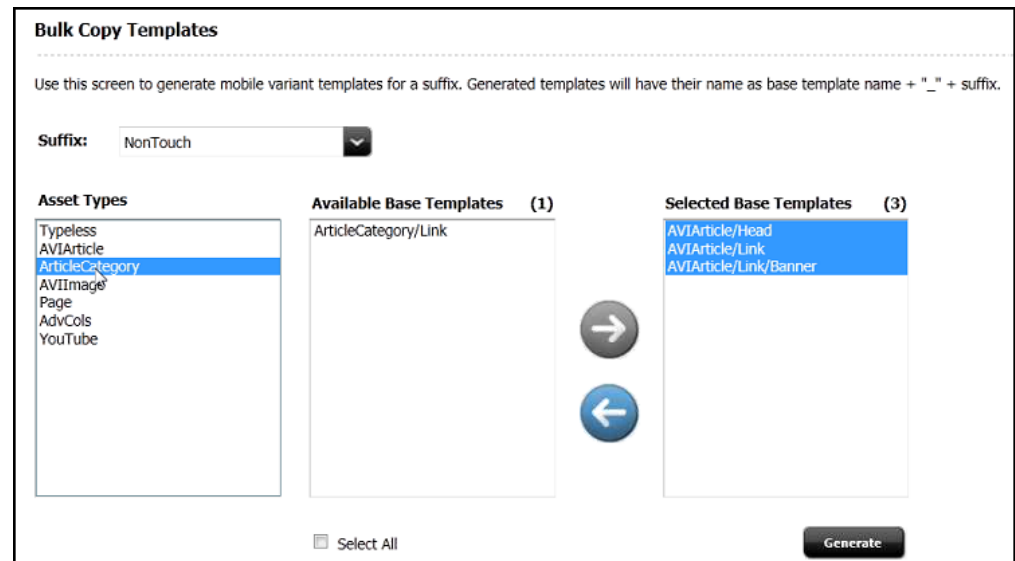
5. Click **Generate**.

The **Successfully Generated templates: <name>** message is displayed.

26.7 Optimizing Images for Mobile Websites

The image optimization filter lets you optimize images for your mobile websites.

You can also plug in your own image optimization implementation to optimize images in a custom way. The following sections discuss both these approaches:

- How to Optimize Images Using the Image Optimization Filter

- How to Optimize Images Using a Pluggable Interface

26.7.1 How to Optimize Images Using the Image Optimization Filter

The image optimization feature of WebCenter Sites is available for flex asset families. You can optimize full-size images to display them suitably on various types of mobile devices. This feature enables you to optimize images for different suffixes to support different screen dimensions. For instance, on some devices, the images displayed should be 50% of the full-size images, whereas on other devices, images should be displayed even smaller than 50% of the actual size.

To optimize images using the image optimization filter:

- Create a Flex Filter of the ImageOptimizationFilter Type

- Include the Filter in Your Site's Image Definition

- Create Instances of the blob Type Attribute Asset

- Set Image Properties for Optimization

- Apply the Image Optimization Filter on Existing Images

- Verify If the Image Optimization Filter Has Been Applied

- Use the Optimized Images in Your Site

26.7.1.1 Create a Flex Filter of the ImageOptimizationFilter Type

1. On the menu bar, click **New**.

2. From the list of options, click **New Filter**.

   The New Filter form is displayed.

3. In the **Name** field, enter a name for your new filter.

4. From the **Filter** dropdown list, choose **ImageOptimizationFilter**, then click **Get Arguments**.

Figure 26-25   ImageOptimizationFilter
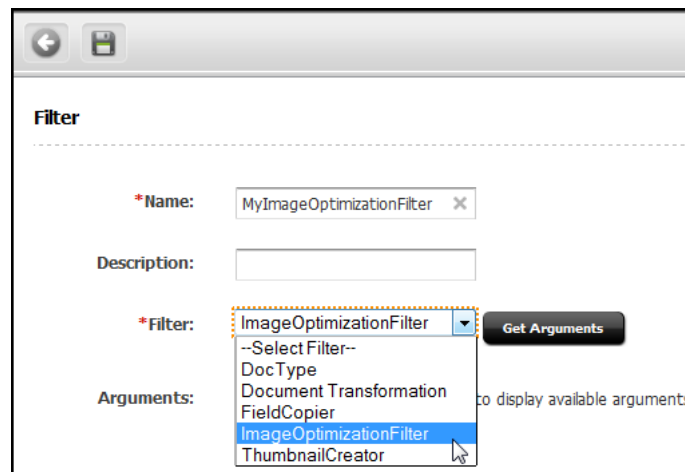
5. Under the **Arguments** section, in the **Value** field, enter the name of the existing `blob` type attribute asset for your site. For example, for the avisports sample site, this asset is `imageFile` and flex family is `content`. Click **Add** to apply this value.

6. Click the **Save** icon.

26.7.1.2 Include the Filter in Your Site's Image Definition

Include the filter you created in step 1 in the image definition for your site (For example, in the avisports sample site, you can search for `Image` definition) by performing the following steps:

1. Open the `Image` definition for editing.

2. In the **Attributes** section, under **Available**, select your flex family, and click **Required**. In avisports, the flex family is already selected.

3. In the **Filters** section, under **Available**, select the image optimization filter created in the previous step (**myImageOptimizationFilter** in our example), and click **Select**.

Figure 26-26   Image Definition
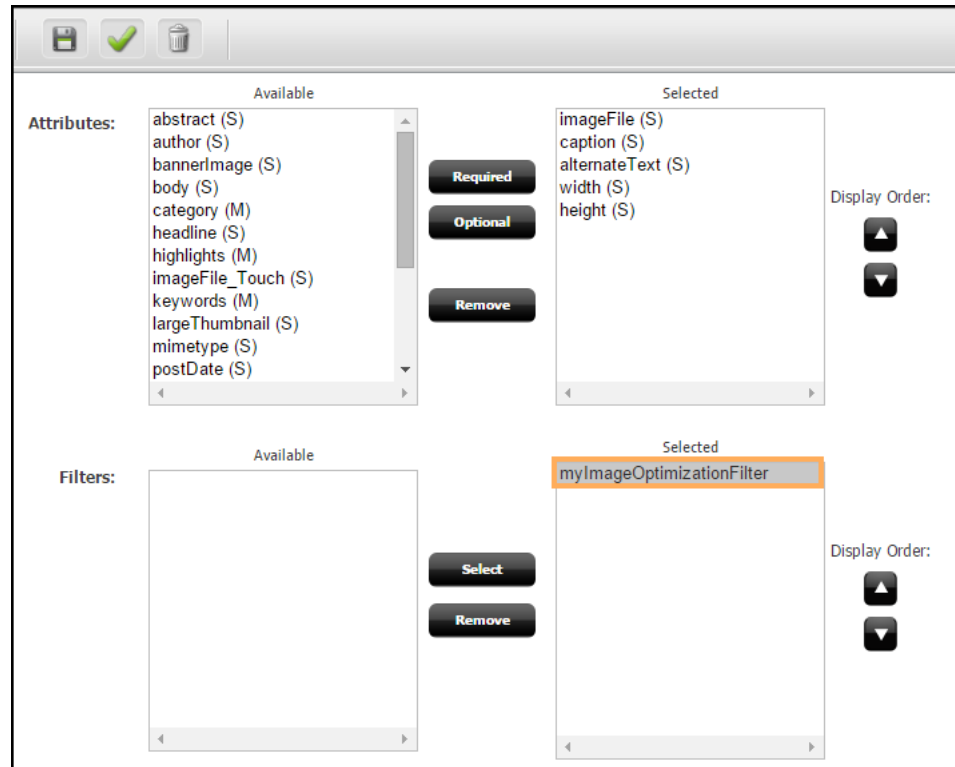
4. Click the **Save** icon.

26.7.1.3 Create Instances of the blob Type Attribute Asset

In WebCenter Sites, both single-valued and multi-valued blob type attributes are supported. To apply the image optimization filter to all template variants, create as many instances of the `blob` type attribute asset as there are suffixes for your site. For example, in avisports sample site, the existing `blob` type attribute is `imageFile`. So, in this case, create an instance of this attribute asset for the `Touch` suffix and another for the `NonTouch` suffix:

1. On the menu bar, click **New**.

2. From the list of options, click **New Attribute**.

    The New Attribute form is displayed.

3. In the **Name** field, enter a name in format `<blob_attribute_name>_<Suffix>`. So, for avisports site, it would be `imageFile_Touch` and `imageFile_NonTouch`.

4. From the **Attribute Type** dropdown list, choose **blob**.

5. Click the **Save** icon.

26.7.1.4 Set Image Properties for Optimization

1. Under the **Mobility** tab, double-click **Image Properties** node.

2. For the required suffixes, set the following properties:

- `preferredFormats`: Optional property. Different types of image formats. For example, for avisports site, for both `Touch` and `NonTouch` suffixes, the formats are: `jpg,png,bmp,gif`. The purpose of various formats is to get the smallest image. The formats are provided in the order of priority. If the first format `jpg` can give an optimized image, then this format is used. Otherwise, the format with next priority is used. In some cases where none of the supported formats may be helpful in achieving the required size, the full-size image is rendered on devices.

  When omitted, the format of the optimized image is the same as the original/full size image.

- `targetSize`: The average size of images on devices. This must be in percentage. For example, 40% of the full-size image.

- `maxsize`: The maximum size of images on devices. This must be in percentage. For example, 50% of the full-size image. The optimized image that exceeds the `maxsize` is not stored as a rendition. If the size (in bytes) of the optimized image exceeds `maxSize`, the main image is used.

26.7.1.5 Apply the Image Optimization Filter on Existing Images

The image optimization filter comes into play when an image is created or edited. So, to apply this filter on existing images, they must be edited and saved.

1. Open the images in Edit mode.

2. Click the **Save** icon to apply the filter.

26.7.1.6 Verify If the Image Optimization Filter Has Been Applied

1. Open the Contributor interface.

2. Search and open an image.

   The Content tab of the image contains a thumbnail for each suffix. The figure below shows the Content tab of a sample image with three thumbnails: one for a full-size image (for the Default site), one for `Touch` suffix, and one for `NonTouch` suffix.

Figure 26-27   Sample Image Content
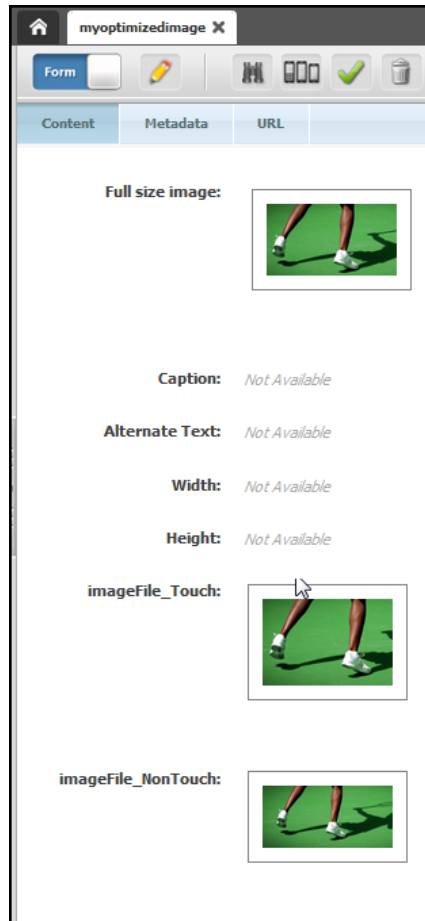
3. To view the optimized image for each suffix, click the respective URLs available on the URL tab.

Figure 26-28   Sample Image's URLs
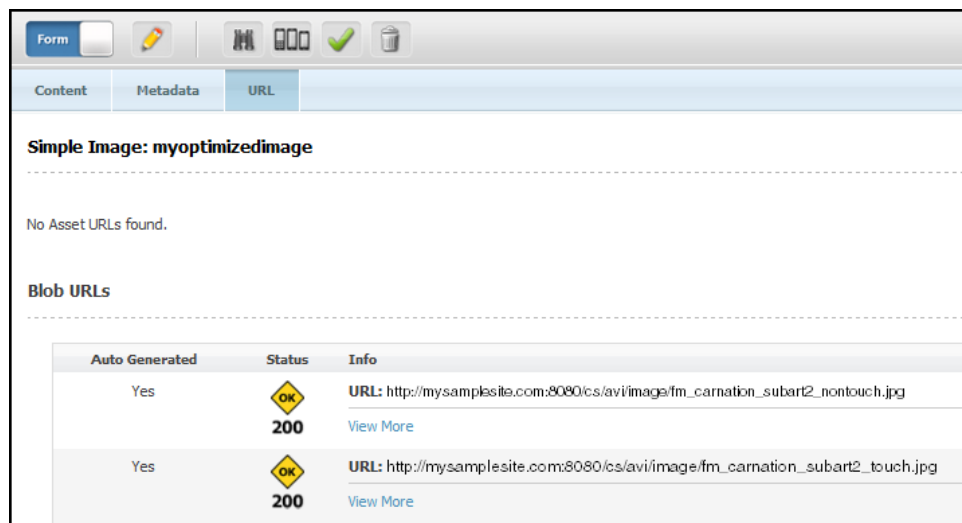
26.7.1.7 Use the Optimized Images in Your Site

1. Set the `optimize` attribute of the `render:getbloburl` tag to `true` (`<render:getbloburl c=... cid=.. optimize='true' />`). See *Oracle Fusion Middleware Property Files Reference for Oracle WebCenter Sites*.

26.7.2 How to Optimize Images Using a Pluggable Interface

To optimize images for your mobile websites, plug in a custom implementation of the image optimization API, `ImageOptimizer.java`, as follows:

1. Create a class that extends abstract class `ImageOptimizer.java`. For example, see the default implementation of the `ImagePercentScaler.java` class included in WebCenter Sites: `com.fatwire.mobility.image.impl.ImagePercentScaler`.

2. Optionally, create a class that extends the abstract class `TargetImageProperties.java`. For example, see the default implementation of the `DefaultTargetImageProperties.java` class included in WebCenter Sites: `com.fatwire.mobility.image.impl.DefaultTargetImageProperties`. Or, reuse the existing default implementation class itself if properties other than those allowed in default implementation are not used.

   > **Note:**
   >
   > If you use the default target properties, do not change the first line in the XML snippet shown in step 3.

3. In the `MobilityService.xml` configuration file located at `<WebCenter Sites_HOME>/config/`, replace the value of the attribute class with the respective implementation of the `TargetImageProperties.java` and `ImageOptimizer.java` abstract classes (`COM.FutureTense.Mobility.Image.ImageOptimizer` and `COM.FutureTense.Mobility.Image.TargetImageProperties`).

```
<bean id="TargetImageProperties"
class="com.fatwire.mobility.image.impl.DefaultTargetImageProperties"
singleton="false"/>
<bean id="ImageOptimizationService"
class="com.fatwire.mobility.image.impl.ImagePercentScaler" singleton="true" >
      <constructor-arg ref="TargetImageProperties"/>
</bean>
```

Now you are ready to use your implementation. For a similar implementation, see the following code:

```
ImageOptimizer srv = ServiceLocator.getService( "ImageOptimizationService",
ImageOptimizer.class );
TargetImageProperties targetImageProperties = srv.getTargetImageProperties();
 targetImageProperties.putAll( customPropertnValues );
 //'customPropertnValues' is a map that contains custom property names and values
as required by userimplementation ofimageoptimization API
```

26.8 How Device Detection Works

WebCenter Sites uses a built-in device detection mechanism to identify the device that requests website content. Once the device is identified, WebCenter Sites looks for the

matching device group and reads its suffix. Using the suffix, it finds the site navigation and invokes the template variant to render the content.

The detailed steps are as follows:

1. Remote Satellite Server receives a page request from a real device. The header for this request includes the user agent of the device.

2. Remote Satellite Server looks for the page in its own cache. If it fails to find the page, Remote Satellite Server sends the page request to WebCenter Sites.

3. WebCenter Sites responds as follows:

   a. Identifies the device by its user agent in the request header.

   b. Looks for the user agent in the device repository.

   c. If it finds a matching device in the device repository, WebCenter Sites also looks for the capabilities of that device.

   d. WebCenter Sites then uses the user agent and device capabilities to find device groups with matching criteria.

   e. Associates the device to the highest-priority device group.

   f. Reads the suffix for this device group.

   g. Assigns the suffix to the `d` parameter in the `ics` scope.

   h. Appends `_suffix` to the requested template name in the URL.

   i. Appends `d=suffix` to the URL of the requested page.

      For example:

      If the suffix is `Touch`, WebCenter Sites converts the original URL `pagename=avisports/HomeLayout1&c=Page&cid=1482760932` to the following URL: `pagename=avisports/HomeLayout1_Touch&c=Page&cid=1482760932&d=Touch`

   j. If the template variant exists, WebCenter Sites executes the new URL, caches the page, and sends it to Remote Satellite Server.

      If the template variant does not exist, WebCenter Sites executes the original URL, caches that page, and sends it to Remote Satellite Server.

4. Remote Satellite Server caches the page and sends the response back to the device.

5. Remote Satellite Server also caches device detection information and uses it to process subsequent requests from the same device. This prevents WebCenter Sites from re-running device detection.

Part V

Coding with Developer Tools

Developer Tools, a toolkit used to integrate Oracle WebCenter Sites with the Eclipse Integrated Development Environment (IDE), lets you work in a distributed environment using the Eclipse IDE and version control systems (VCS).

Topics:

- About Developer Tools

- Installing and Configuring Developer Tools

- Introducing Developer Tools Features in Eclipse

- Developing JSPs with Developer Tools

- Creating Templates for Mobile Websites Using Developer Tools

- Synchronizing and Exchanging Data Using Developer Tools

- Using Workspaces in Developer Tools

- Using Developer Tools Command Line Interface (CLI)

- Integrating Developer Tools Workspaces with Version Control Systems

- Using Developer Tools to Manage and Exchange Resources

- Using the Developer Tools Command Line Interface (CLI) to Create Reusable Modules

27

About Developer Tools

Oracle Developer Tools lets you integrate WebCenter Sites with the Eclipse IDE so you can create a personal and flexible development environment.

Topics:

27.1 Introduction to Developer Tools Architecture

You and other developers like you would like to work in personal and flexible environments. Integrate WebCenter Sites with the Eclipse IDE to create an environment that improves your productivity. You interact with Developer Tools (and therefore WebCenter Sites), primarily through Eclipse. And, you get a rich set of functions for managing WebCenter Sites resources.

Developer Tools may be running on any computer (local or remote). Whether managed through Eclipse or in WebCenter Sites, the resources can be either automatically or manually synchronized by the Developer Tools kit.

For example, Eclipse-managed resources are stored as files in a file system, giving developers the option to integrate with a version control system of their choice. If the resources are modified and WebCenter Sites is running, then the resources are automatically synchronized, that is, imported into WebCenter Sites, in its native database representation. Manual synchronization can be performed in both directions.

This figure shows a summary of Developer Tools:

Figure 27-1   Developer Tools Process Flow
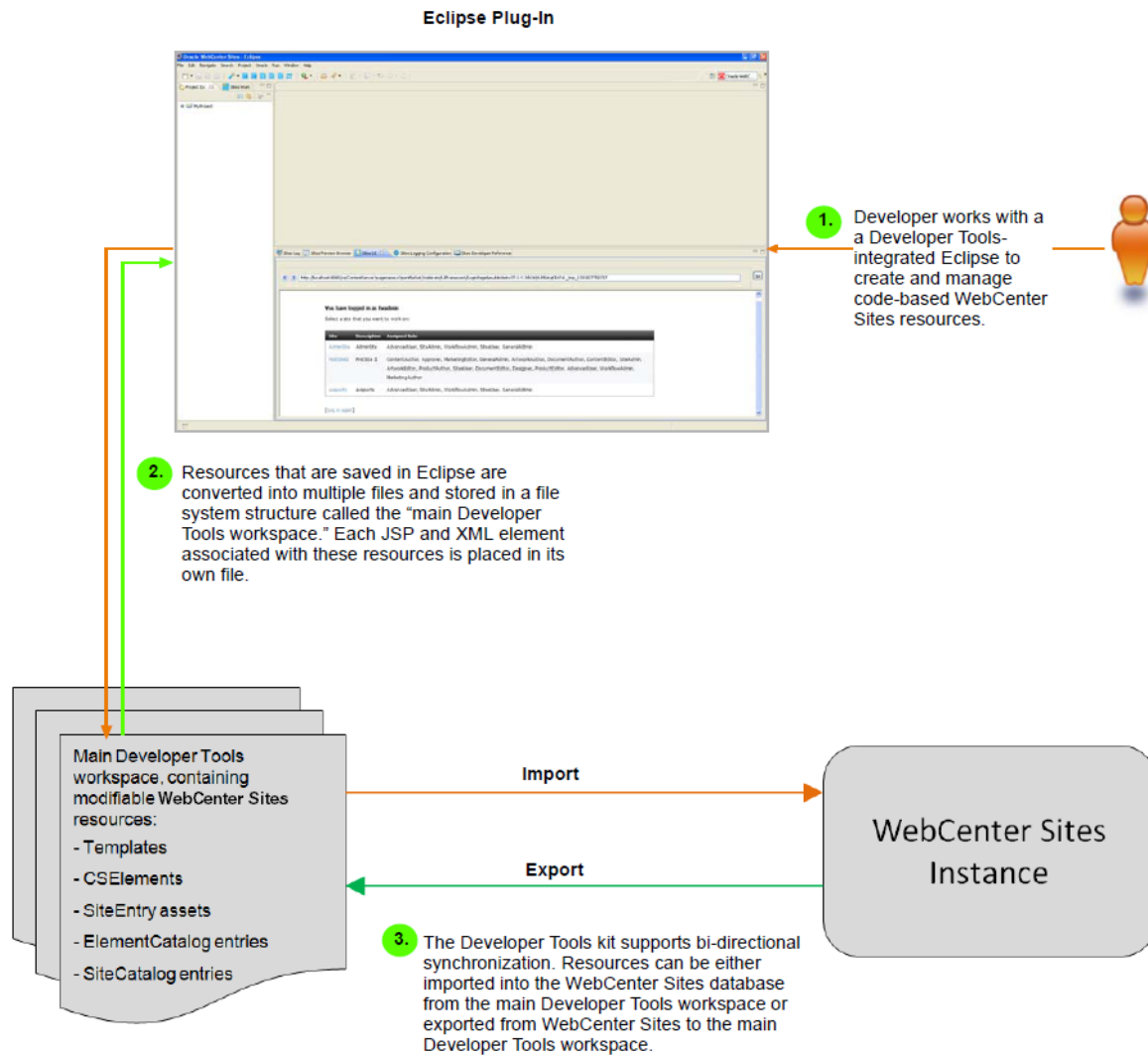
27.2 IDE Integration

With Eclipse, you can create and manage templates and elements, export and import assets, preview pages, and so on.

- Create, edit, and delete CSElement, Template, Controller, SiteEntry assets, and SiteCatalog and ElementCatalog entries.

- Develop JSP elements with standard Eclipse features such as tag completion, syntax highlighting, and debugging.

- Export and import assets, asset types, flex families, sites, roles, tree tabs, and start menu items.

- Preview WebCenter Sites pages within the Eclipse IDE using an embedded preview browser.

- View the WebCenter Sites log file in a dynamically refreshing panel.

- Integrate with version control systems.

> **Note:**
>
> Integrating Developer Tools with Eclipse embeds the Admin and Oracle WebCenter Sites: Contributor interfaces in Eclipse to make them easily accessible to developers. The Admin interface is used in many parts of this guide (for example, to create flex families). See What Can You Do in the Contributor Interface? in*Oracle Fusion Middleware Using Oracle WebCenter Sites*.

27.3 The Developer Tools Workspace

WebCenter Sites resources that you manage in Eclipse are stored as files in a file system structure called the *main Developer Tools workspace*.

This structure enables resources to be easily managed and exchanged with WebCenter Sites instances. The main Developer Tools workspace is the only workspace accessible from Eclipse.

> **Note:**
>
> Creating a custom workspace is optional. In most distributed environments the only necessary workspace is the main Developer Tools workspace. Custom workspaces are not accessible from Eclipse. Creating a custom workspace is described in Using Workspaces in Developer Tools. All other Developer Tools topics in this guide discuss the main Developer Tools workspace.

27.4 Connecting to WebCenter Sites Instances

Developer Tools enables you to connect to any local or remote WebCenter Sites instance. Each WebCenter Sites instance that you integrate with Eclipse is assigned an Eclipse project and each project is displayed in the main Developer Tools workspace. The Eclipse project tracks the WebCenter Sites resources stored in Eclipse for that WebCenter Sites instance.

You can connect to as many WebCenter Sites instances (locally and remotely) as you want; an Eclipse project folder is created for each instance you connect to. See Understanding Projects and Workspaces in Eclipse.

27.5 Synchronization

With the Developer Tools kit, you synchronize resources in Eclipse with those in WebCenter Sites. This way you ensure that the Developer Tools workspace and WebCenter Sites database contain the same content.

Manual synchronization is bi-directional; that is, you can import resources into the WebCenter Sites database and export resources to the Developer Tools workspace.

Exporting resources to Eclipse converts the resources into files. Importing a resource into WebCenter Sites converts the resource to native WebCenter Sites format (database representation).

> **Note:**
>
> Automatic synchronization occurs when WebCenter Sites resources are edited, created, or deleted in Eclipse, but only if the WebCenter Sites instance is running. This synchronization includes transparent flushing of page and resultset caches in WebCenter Sites.

27.6 JSP Management

The Developer Tools kit exposes WebCenter Sites JSPs as individual files in the Developer Tools workspace for you to manage the JSPs as any other JSP file.

That is, you can create, edit, and debug your JSPs as any other JSP file.

27.7 Command Line Interface (CLI)

With the Developer Tools kit's command line interface you can automate import and export tasks and large-scale resource movement. You can also create custom workspaces, and synchronize resources with any workspace.

The Eclipse integration lets you work only with resources stored in the main Developer Tools workspace.

27.8 About Using a Version Control System

A version control system lets you check out resources to any target system, including testing servers, Management WebCenter Sites systems, or another developer's WebCenter Sites instance, to make use of their functions (such as publishing). Your Developer Tools kit supports version control system. So, you can implement one to be able to exchange resources between WebCenter Sites instances and your colleagues. You also can update your workspace with the resources checked in by other developers.

The figure shows an example of using Developer Tools with a version control system. This example uses a dedicated WebCenter Sites instance to publish resources to a Management WebCenter Sites instance. Therefore, the Approval/Publishing feature provided by WebCenter Sites can be used to publish resources that were checked out from the version control system. This example is the recommended way to use a version control system with Developer Tools, but it is not required.

Figure 27-2 Using Developer Tools with a Version Control System

28
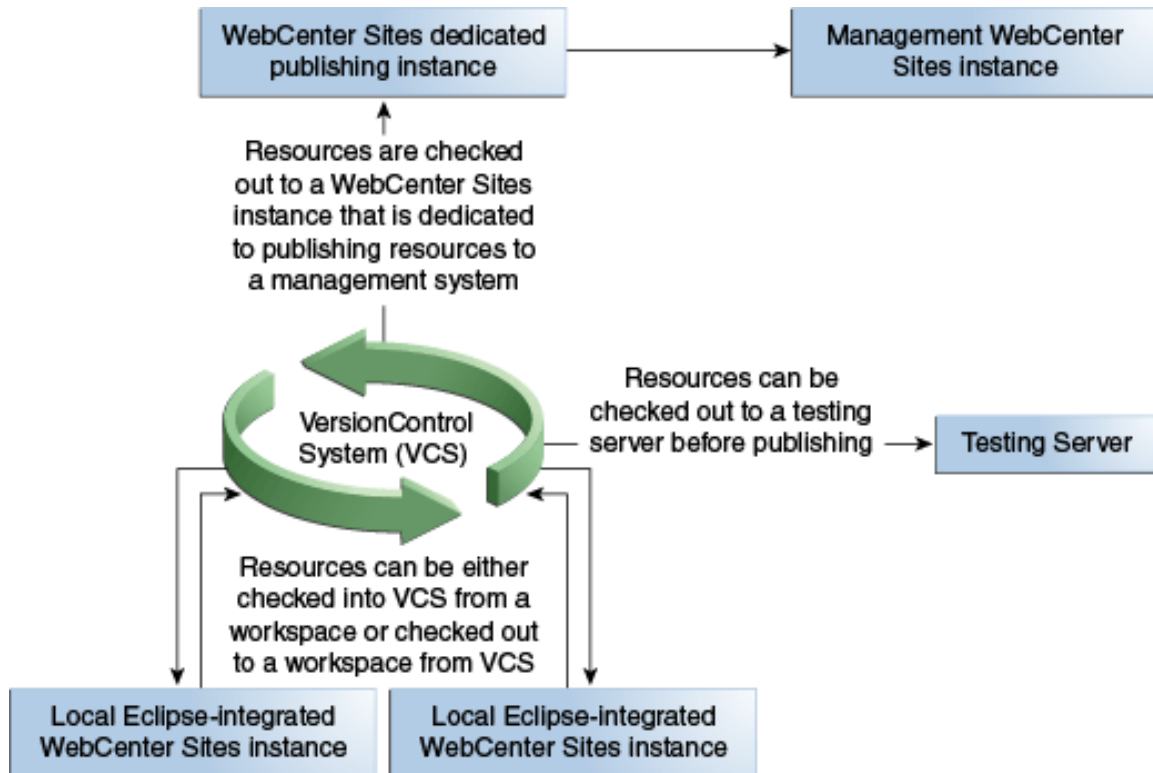
Installing and Configuring Developer Tools

Would you like to start managing WebCenter Sites resources in Eclipse? Set up Developer Tools on your machine and integrate your instance of WebCenter Sites with Eclipse.

For information about installing, configuring, and updating Developer Tools, as well as managing WebCenter Sites resources in Eclipse, see these topics:

- Prerequisites

- Setting Up Developer Tools

- Updating Developer Tools

- Managing WebCenter Sites Resources in Eclipse

- Uninstalling Developer Tools

28.1 Prerequisites

Are you planning to set up Developer Tools on your system? Here are some important points that you should keep in mind before you set up Developer Tools.

- The Developer Tools plug-in you need to integrate with the Eclipse IDE and the executable JAR file for the command line interface are both located in the `${Oracle Home}/wcsites/clients` directory of your WebCenter Sites installation.

- The Developer Tools and Eclipse IDE work with JDK 1.8 and higher versions only. If you use JDK versions lower than 1.8, then the Eclipse IDE plugin reports that it is running on a lower platform than specified, and this might cause code runtime errors due to an unmatched platform.

- Determine whether you will be connecting to a local or remote WebCenter Sites instance.

- After you have integrated Eclipse with WebCenter Sites, you must log in to WebCenter Sites with general administrator credentials (for example, `fwadmin/xceladmin`). This user must be a part of the `RestAdmin` group.

- To use the command line interface feature:

  - Developer Tools requires access to the CAS internal URL for the WebCenter Sites instance to which you are connecting. The CAS internal URL is specified during the Oracle WebCenter Sites installation. The internal URL must be publicly accessible for Developer Tools to access it.

    See Configuring the CAS Primary Cluster Node in *Oracle Fusion Middleware Installing and Configuring Oracle WebCenter Sites*.

– You must have an advanced knowledge of Developer Tools. See Using Developer Tools Command Line Interface (CLI).

28.2 Setting Up Developer Tools

To set up Developer Tools, you install the Developer Tools Plug-in, integrate WebCenter Sites with it, and enable code completion on remote hosts. You also need to know how to work with existing resources using Developer Tools and how to manage resources.

- How to Install the Developer Tools Plug-in

- How to Verify the Developer Tools Plug-In Installation

- How to Integrate WebCenter Sites with the Eclipse IDE

- How to Enable Code Completion for Remote Hosts

- How to Use Developer Tools to Work with Existing Resources

- How to Manage WebCenter Sites Resources

- How to Work with a Pre-Existing Project in Eclipse

28.2.1 How to Install the Developer Tools Plug-in

1. Download the Eclipse package that suits your development needs. The Eclipse IDE for Java EE Developers is recommended for development on WebCenter Sites. You can download Eclipse from the following URL:

   http://www.eclipse.org/downloads/

2. Locate the `clients` folder in your WebCenter Sites installation directory (`${Oracle Home}/wcsites/clients`). This folder contains the following zip files:

   - `com.oracle.sites.developer-tools-xyz.zip`: Developer Tools plug-in.

   - `developer-tools-command-line.zip`: Command line interface zip file, which contains the executable JAR file for the command line interface.

3. Start Eclipse (run `eclipse.exe`).

   Note:

   Eclipse installs plug-ins from local file system (or archive) locations and remote locations. The location of a plug-in is referred to as a *software site*. For the Developer Tools plug-in, the software site is the path to the Developer Tools plug-in on the file system.

   ---

4. From the Eclipse menu, choose **Help**, then choose **Install New Software**.

   The installation wizard opens showing **Available Software**).

5. On the Available Software page, click **Add** (located to the right of the **Work with** field).

6. In the Add Repository dialog, fill in the following fields, and then click **OK**.

   - In the **Name** field, enter a unique name for the Developer Tools plug-in.

   - In the **Location** field, click **Archive** to specify the location (software site) of the Developer Tools plug-in located in the `client` folder.

   The installation wizard picks up the Developer Tools plug-in from the specified location (software site), and lists the name of the plug-in on the Available Software page.

7. On the Available Software page, select **Oracle WebCenter Sites Developer Tools**, then click **Next** .

Figure 28-1   Installation Wizard: Available Software Listing the Developer Tools Plug-In
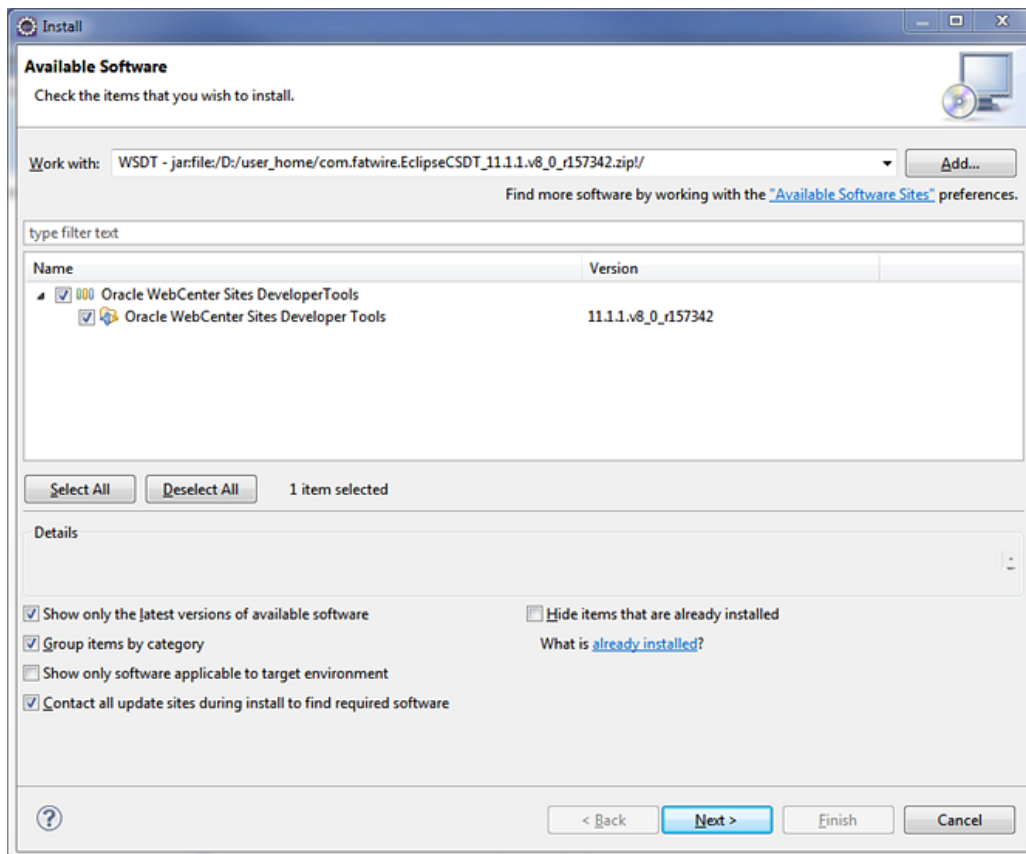
The Install Details page opens.

8. On the Install Details page, click **Finish** .

Figure 28-2   Installation Wizard: Install Details Window
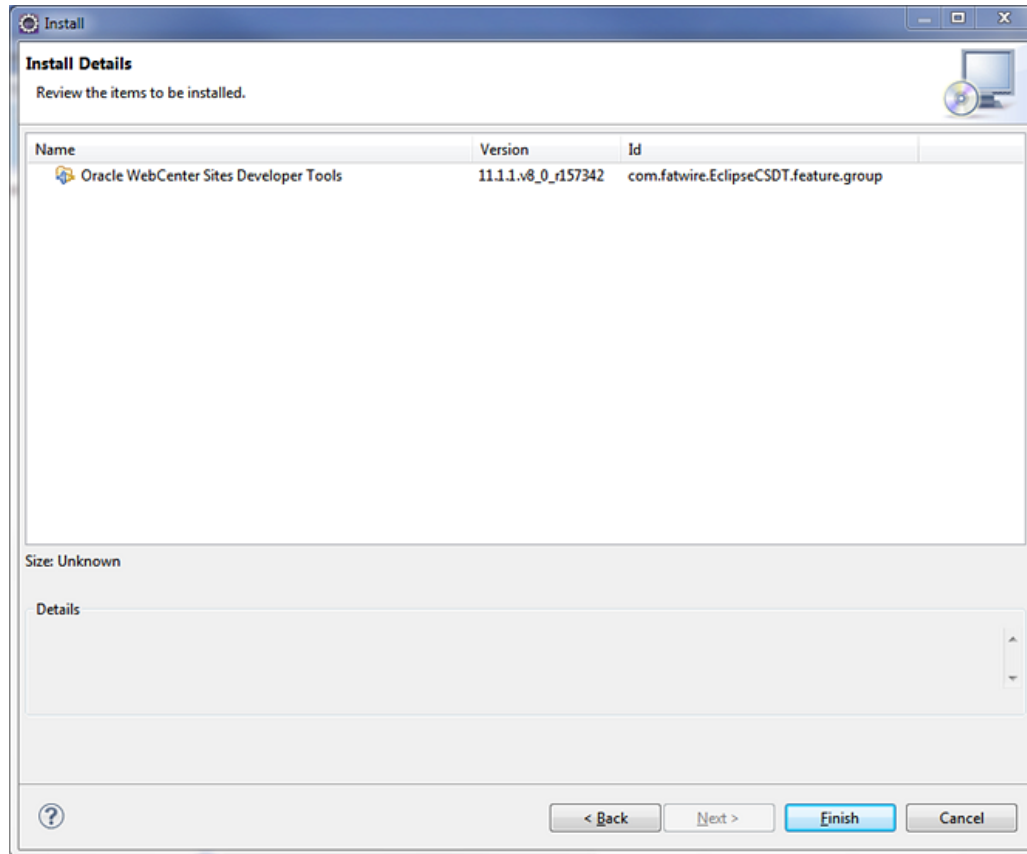
9. During the installation process, a Security Warning dialog opens. Click **OK** to continue with the installation.

10. After the installation completes successfully, the Software Updates dialog open.

    To ensure WebCenter Sites is running, access `${base_url}/HelloCS`, where `${base_url}` is the base URL for your WebCenter Sites instance. For example, if your base URL is `http://example:8080/cs`, then `http://example:8080/cs/HelloCS` is the verification URL for your WebCenter Sites instance.

11. Restart Eclipse by clicking **Yes** in the Software Updates dialog.

    Restarting Eclipse refreshes the plug-in cache and makes the Developer Tools plug-in available in Eclipse.

28.2.2 How to Verify the Developer Tools Plug-In Installation

1. After Eclipse restarts, verify that the Developer Tools plug-in is installed successfully. From the **Eclipse** menu, choose **Help**, then choose **About Eclipse**.

    The About Eclipse dialog opens showing the **Developer Tools** icon in the list of installed plug-ins.

Figure 28-3   About Eclipse Dialog

2. Open the Oracle WebCenter Sites perspective by clicking the **Open Perspective** icon, located on the top right of the Eclipse IDE. In the Open Perspective dialog, select **Oracle WebCenter Sites**, then click **OK**.

Figure 28-4   Open Perspective: Oracle WebCenter Sites
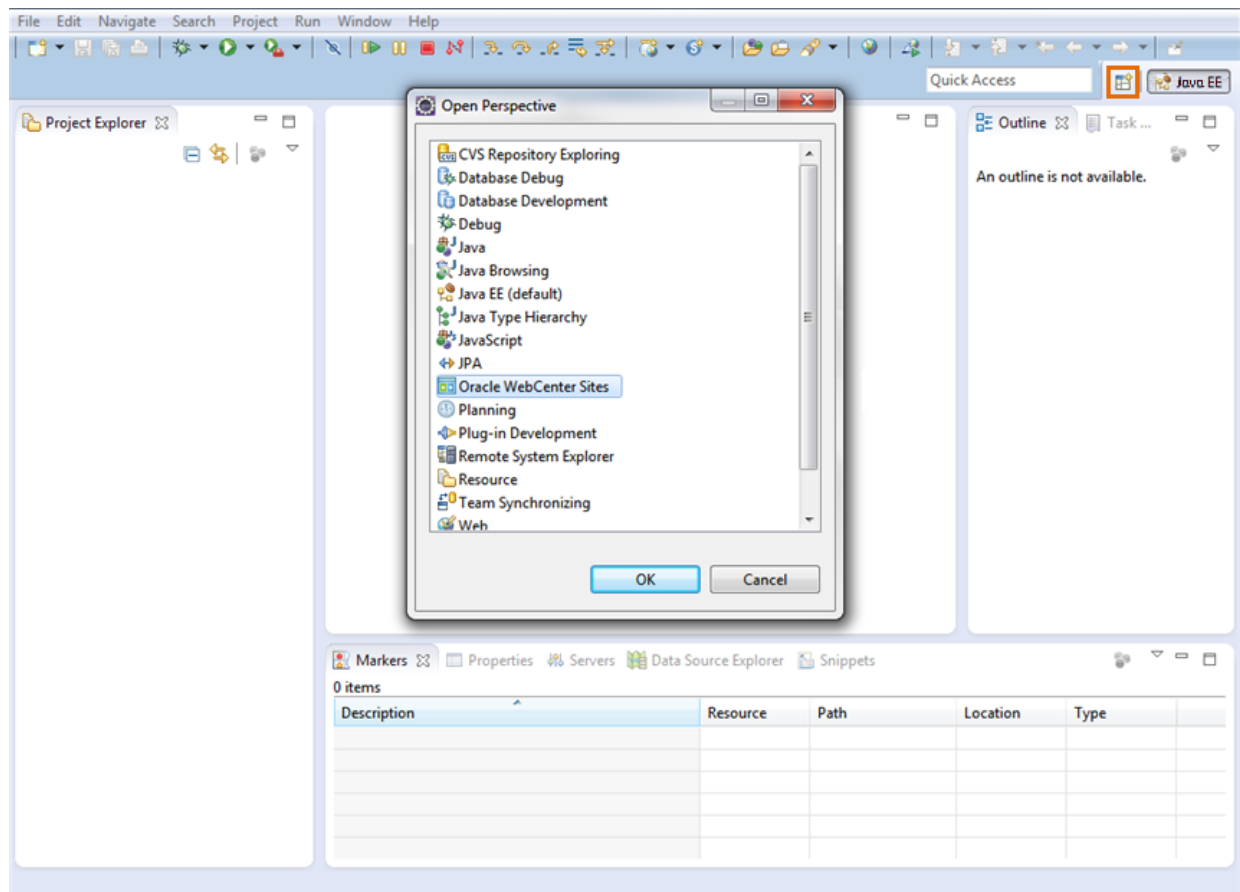
28.2.3 How to Integrate WebCenter Sites with the Eclipse IDE

If you are setting up Developer Tools for the first time, the configuration form is automatically displayed.

If you have connected to a WebCenter Sites instance and want to connect to a different instance, navigate to the WebCenter Sites toolbar and click the **ConfigurePreference** wizard to open the configuration form.

To integrate WebCenter Sites with the Eclipse IDE, enter information for the WebCenter Sites instance to which you want to connect into the configuration form, as follows:

Figure 28-5   Configuration Form
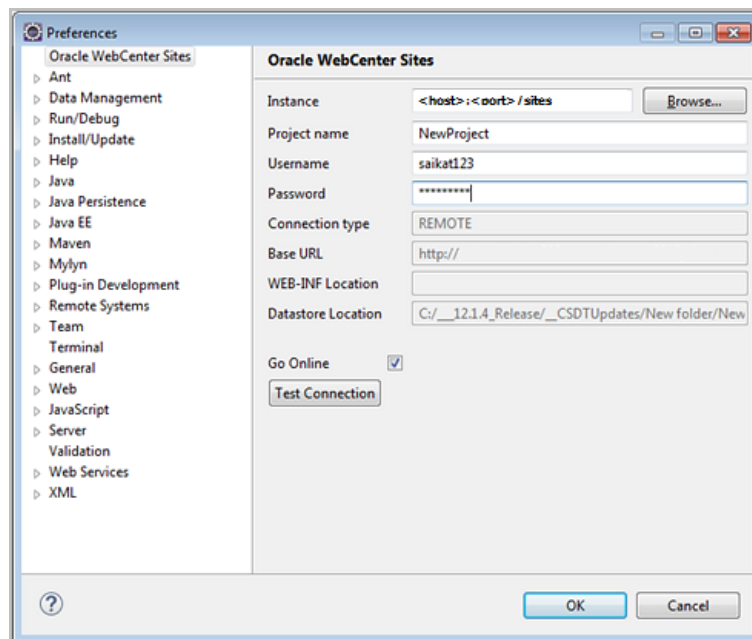
1. In the **Instance** field, do one of the following, depending on whether you are connecting to a local or remote host:

   > **Note:**
   >
   > A remote connection supports the creation of multiple projects on the same remote host. Local connections support only one project at a time.

   - If you are connecting to a local host, click **Browse** to select the locations of the `config` folder (which contains the `wcs_properties.json` file for the WebCenter Sites instance).

   - If you are connecting to a remote host (for example, `http://example:8080/cs`), enter the path to the host in the format `[host]:[port]/[path]`. (You must prefix either `http://` or `https://`.)

   After you enter a value in the **Instance** field, the following fields are automatically populated. The values of these fields are based on whether you specified a local or remote host in the **Instance** field:

- **Remote Connection**: If you are connecting to a remote host, the value is set to `true`. If you are connecting to a local host, the value is set to `false`.

- **Host IP Address**: Specifies the IP address of the host to which you are connecting.

- **Port**: Specifies the port number of the host to which you are connecting.

- **Web Context Path**: Specifies the context path to the host.

- **Workspace**: Specifies the location of the Developer Tools workspace (Eclipse project).

- **Sites Context Root**: If you are connecting to a local host, this field contains the path of the WebCenter Sites web application. If you are connecting to a remote host, this field is blank.

2. In the **Project name** field, enter a name for the project on which you will be working.

3. In the **Username** field, enter the user name of a general administrator. This user must be a member of the `RestAdmin` group.

4. In the **Password** field, enter the password for the user name.

5. Select the **Go Online** checkbox to access the server and work with the WebCenter Sites resources in Eclipse.

6. Click **Test Connection**.

   A dialog opens stating that the connection was successful.

7. In the dialog box, click **OK**.

8. In the configuration form, click **OK**.

   The Oracle WebCenter Sites perspective opens. The following figure shows how the Oracle WebCenter Sites perspective look if you are accessing WebCenter Sites-integrated Eclipse for the first time.

Note:

You are not required to open the **Oracle WebCenter Sites** perspective to work with WebCenter Sites resources.

Figure 28-6   Oracle WebCenter Sites Perspective
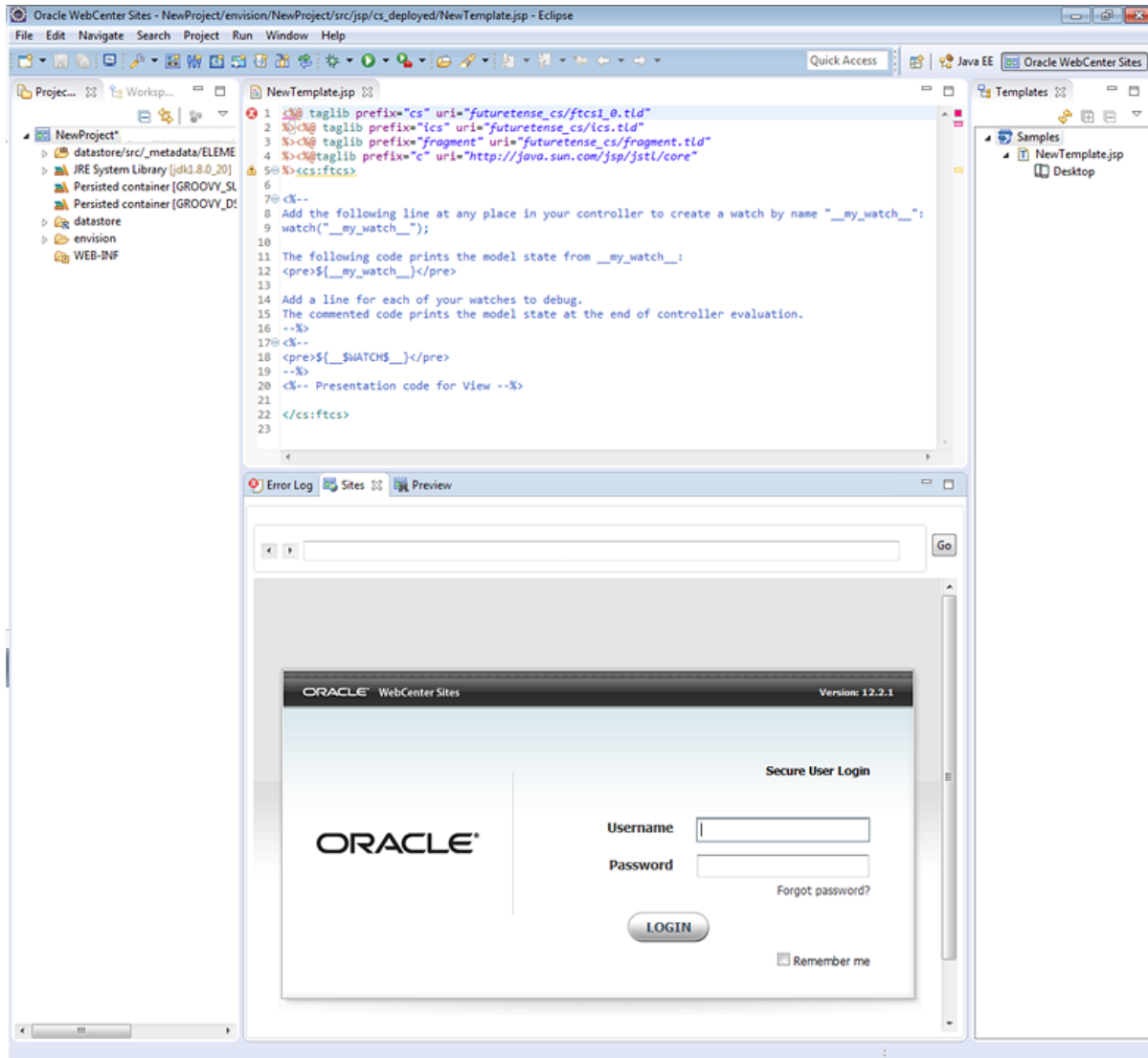
If the Oracle perspective renders as shown in the figure above, then you have successfully set up Developer Tools. When WebCenter Sites is integrated with Eclipse, you are not required to open the **Oracle WebCenter Sites** perspective to work with WebCenter Sites resources.

Note:

The panels containing the Eclipse and Developer Tools views are interchangeable. To move a view to a different panel, click the view's tab and drag it to the panel.

28.2.4 How to Enable Code Completion for Remote Hosts

1. Copy the `WEB-INF` directory from the remote computer to your local file system. It is recommended that you copy it into your Eclipse project folder, located under the workspace folder you created when you initially integrated Eclipse with WebCenter Sites.

2. In the Project view in Eclipse, right-click the `WEB-INF` folder and select **Properties**.

   The Properties for WEB-INF window opens.

   Figure 28-7   Properties for WEB-INF Window
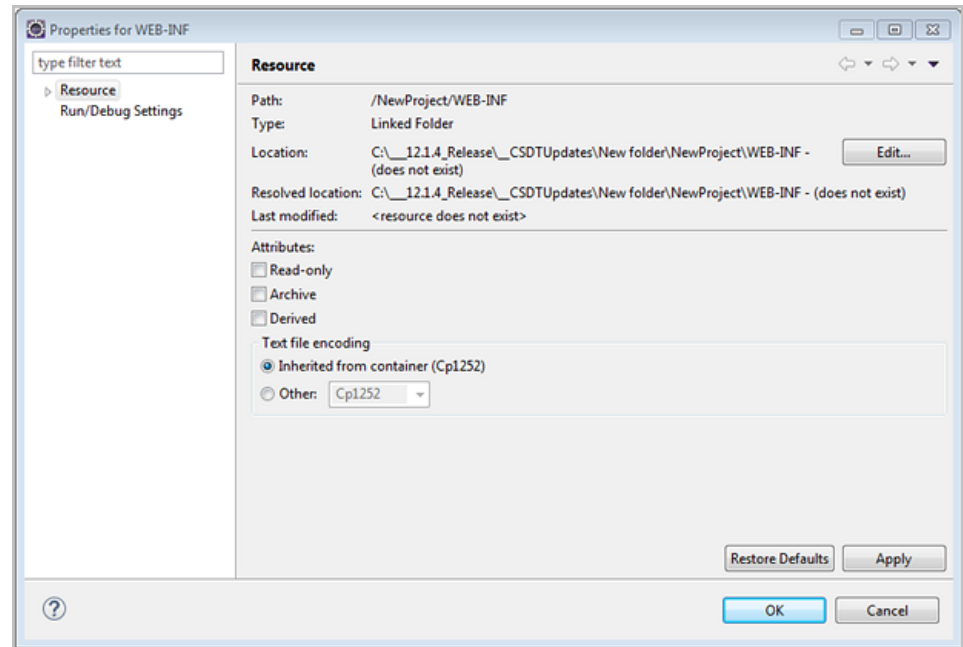
3. In the **Location** field, click **Edit**.

   The Edit a Link Location window opens.

   Figure 28-8   Edit a Link Location Window
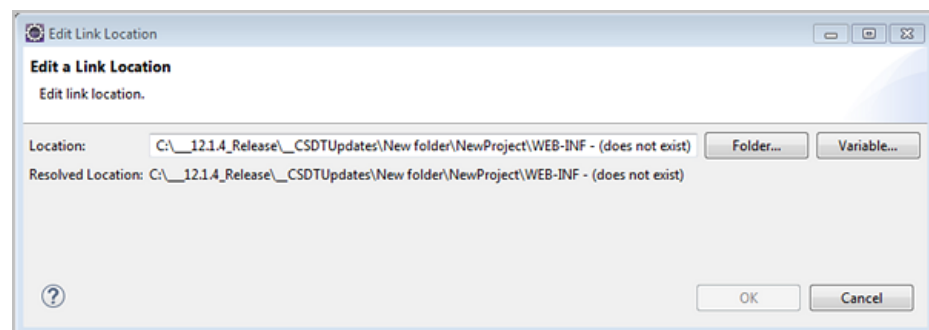
4. In the **Location** field, click **Folder...** and then navigate to the location of the `WEB-INF` folder you copied from step 1. Then, click **OK**.

5. In the Properties for WEB-INF window, click **OK**.

   This enables Eclipse to perform tag completion and proper syntax highlighting.

   Note:

   Code completion is enabled automatically for local instances.

28.2.5 How to Use Developer Tools to Work with Existing Resources

If you upgraded your WebCenter Sites system and want to use Developer Tools to work with resources created before this release (existing resources), see What You Should Know About Using Developer Tools with Pre-Existing Resources.

28.2.6 How to Manage WebCenter Sites Resources

To quickly get started with managing WebCenter Sites resources in the Oracle WebCenter Sites perspective, continue to Managing WebCenter Sites Resources in Eclipse.

28.2.7 How to Work with a Pre-Existing Project in Eclipse

When you integrate Eclipse with a WebCenter Sites instance, you specify the name and location of your workspace. Upon subsequent access of this workspace, you must log in with the credentials you specified in the Configuration form when you initially integrated WebCenter Sites with the Eclipse IDE. Otherwise, you will be working offline and have no access to WebCenter Sites resources in Eclipse.

To work with a pre-existing project in the Eclipse IDE:

1. Open the Eclipse IDE and select the workspace associated with the WebCenter Sites instance you want to work with.

   Eclipse opens, however, you are working in offline mode.

2. To go online and gain server access to the WebCenter Sites instance, do one of the following:

   - Select any WebCenter Sites component in Eclipse.

     For example:

     a. Click the **Create New Template** icon.

        An Error dialog opens with a message stating that your current project is currently offline.

     b. Click **Yes**.

        The Credentials dialog opens.

     c. In the Credentials dialog, enter the username and password for the WebCenter Sites instance to which you want to connect, and then click **Go Online**.

   - Open the Configuration form and enter your credentials for the WebCenter Sites instance to which you want to connect, as follows:

     a. In the WebCenter Sites toolbal, click the **ConfigurePreferences** icon.

        The Configuration form opens.

     b. In the Configuration form, enter your credentials into the **Username** and **Password** fields.

     c. Select the **Go Online** checkbox.

Figure 28-9   Configuration Form
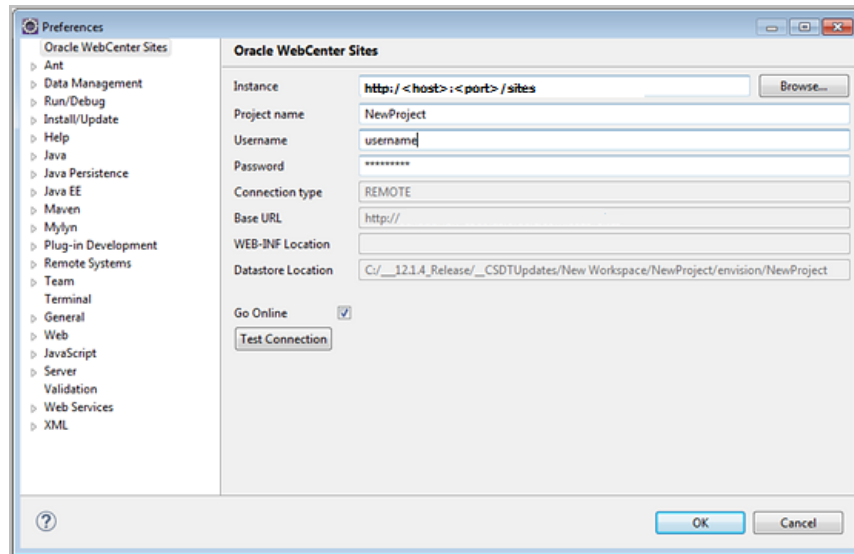
   d.   Click **OK**.

You are now working online and can create and manage WebCenter Sites resources in Eclipse.

28.3 Updating Developer Tools

Here are the steps you need to perform after you've upgraded your WebCenter Sites installation.

- How to Update the Location of the Developer Tools Plug-In

- How to Check for Updates to Existing Plug-Ins

- How To Verify That the Developer Tools Plug-In Has Been Updated

28.3.1 How to Update the Location of the Developer Tools Plug-In

1. Ensure that the updated Developer Tools plug-in
   (`com.oracle.sites.developer-tools-xyz.zip`) is available in the
   `{$Oracle Home}/wcsites/clients` directory.

   Note:

   To ensure you have the latest version of Developer Tools, note the version number (*xyz*) appended to the zip file's name.

   ---

2. Start Eclipse (execute `eclipse.exe`).

3. From the Eclipse menu, choose **Help**, then choose **Install New Software**.

   The Available Software dialog opens.

4. In the Available Software dialog box, click **Add**.

5. In the Add Repository dialog, fill in the following fields, and then click **OK**.

- In the **Name** field, enter a name for the Developer Tools plug-in

- In the **Location** field, click **Archive** to specify the location of the updated plug-in, then click **OK**.

Figure 28-10   Add Repository Dialog
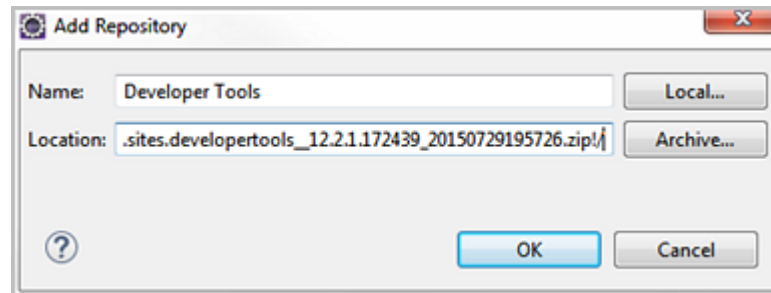
6. In the Available Software dialog box, select the update for the Developer Tools plug-in and then click **Next**.

Figure 28-11   Available Software: Select Updated Plug-in
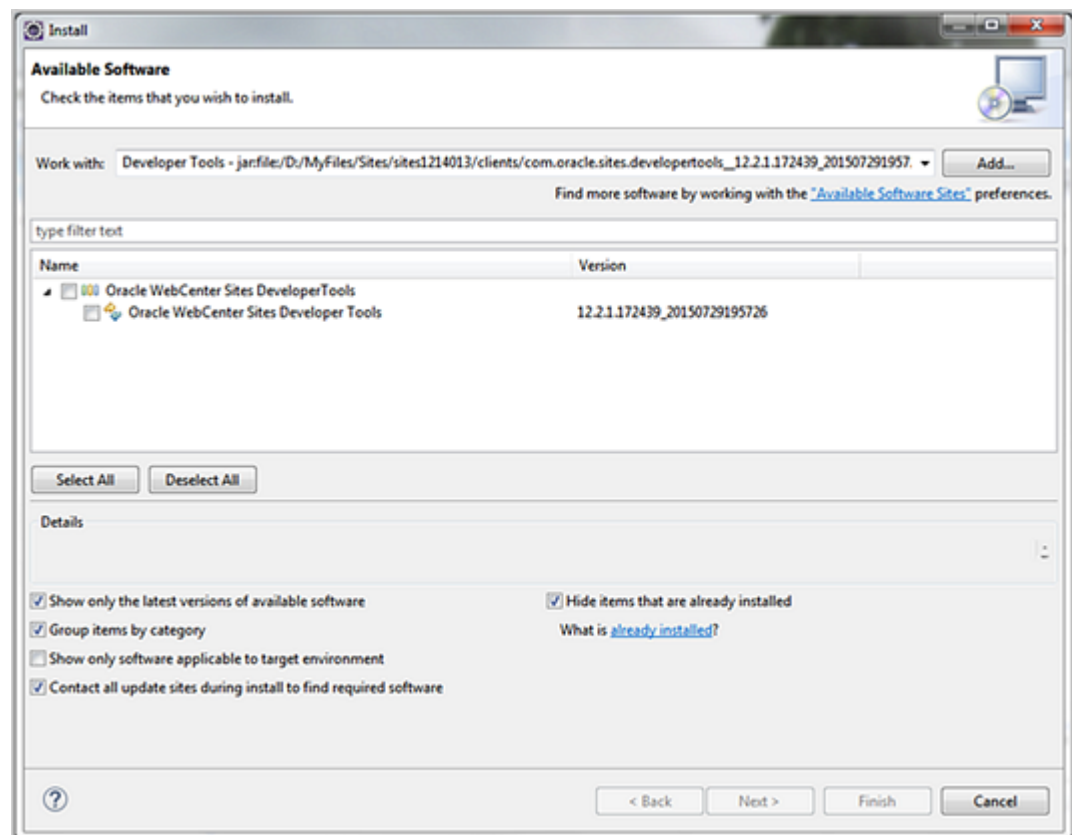
The Install Details dialog opens showing updated plug-in was installed successfully.

7. In the Install Details dialog, click **Finish**.

8. Restart Eclipse for the changes to take effect.

28.3.2 How to Check for Updates to Existing Plug-Ins

1. From the **Eclipse** menu, choose **Help**, then choose **Check for Updates**.

   The Available Updates dialog opens.

2. Select **Oracle WebCenter Sites Developer Tools**, then click **Next**.

   The Update Details dialog opens.

3. Click **Finish**.

4. During the update process, a Security Warning dialog opens. Click **OK** to continue the update.

5. After the update completes successfully, the Software Updates dialog opens. Before proceeding to the next step, start your local WebCenter Sites instance and restart Eclipse by clicking **Yes** in the Software Updates dialog.

28.3.3 How To Verify That the Developer Tools Plug-In Has Been Updated

After Eclipse restarts, verify that the Developer Tools plug-in has been successfully updated to the latest version.

1. From the Eclipse menu, choose **Help**, then choose **About Eclipse**.

   The About Eclipse dialog opens showing the **Developer Tools** icon in the list of installed plug-ins.

   Figure 28-12   About Eclipse Dialog

2. In the About Eclipse dialog, click the **Developer Tools** icon.

   The About Eclipse Features window opens, showing the version details of the Developer Tools plug-in.

Figure 28-13   About Eclipse Features Window
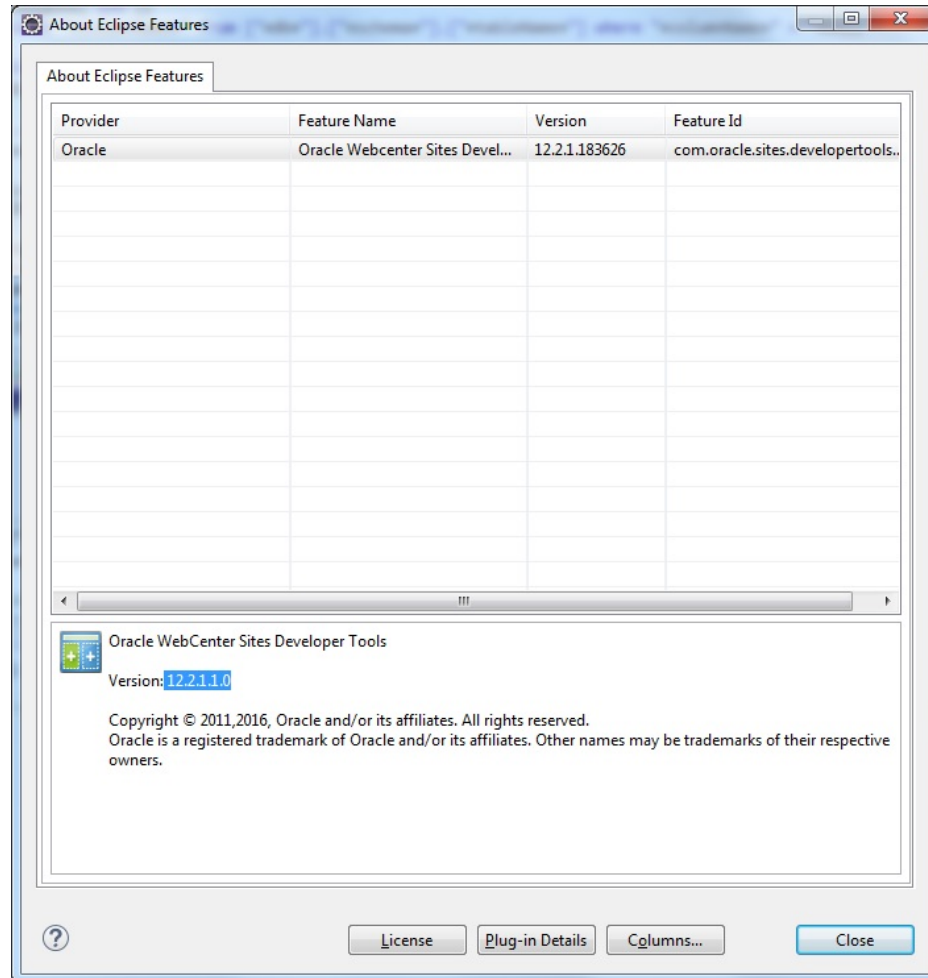
28.4 Managing WebCenter Sites Resources in Eclipse

In Eclipse, you can create, edit, and otherwise manage code-based WebCenter Sites resources.

- Template assets

- Controller assets

- SiteEntry assets

- CSElement assets

- ElementCatalog Entries

- SiteCatalog Entries

See these topics that summarize the steps for managing resources in Eclipse:

- How to Create Resources

- How to Display Developer Tools Views in Panels

- How to Export and Import Data Between WebCenter Sites and Developer Tools

28.4.1 How to Create Resources

1. Start your local WebCenter Sites.

2. Start Eclipse.

3. Open the Oracle WebCenter Sites perspective:

   From the Eclipse menu, choose **Window**, then choose **Perspective**, then **Open Perspective**, then **Other**, and then select **Oracle WebCenter Sites**.

4. Create any or all of the following resources:

   - To create a Template asset, click the **Create New Template** icon and fill in the forms.

   - To create a new Controller asset, click the **Create New Controller** icon and fill in the form.

   - To create a CSElement asset, click the **Create New Element** icon and fill in the forms.

   - To create a SiteEntry asset, click the **Create New Site Entry** icon and fill in the forms.

   - To create an ElementCatalog entry, click the **Create New Element Catalog Entry** icon and fill in the forms.

   - To create a SiteCatalog entry, click the **Create New SiteCatalog Entry** icon and fill in the forms.

   For field definitions and information about Template, CSElement, SiteEntry assets and SiteCatalog and ElementCatalog entries, see Creating Template, CSElement, and SiteEntry Assets. For field definitions and information about Controller assets, see Creating a Controller.

5. When you click **Save**, the resource you created is automatically imported into WebCenter Sites.

6. To manage the resources you create, edit, delete, or share with other sites use the **Sites Workshops Elements** view. Right-click the resource and select an option. For information about the available options, see Workspace.

28.4.2 How to Display Developer Tools Views in Panels

1. Start your local WebCenter Sites.

2. Start Eclipse.

3. Open the Oracle WebCenter Sites perspective:

   From the Eclipse menu, choose **Window**, then choose **Open Perspective**, then **Other**, and then **Oracle WebCenter Sites**.

4. Select **Window**, then **Show View**, and then **Other**.

5. In the Show View dialog, select a view (located under the **Oracle WebCenter Sites** folder):

- **Sites UI** shows the embedded Admin and Oracle WebCenter Sites: Contributor interfaces.

- **Sites Log View** shows the log file for WebCenter Sites.

- **Developer Reference** shows the *Oracle Fusion Middleware Tag Reference for Oracle WebCenter Sites Reference* and *Oracle Fusion Middleware Java API Reference for Oracle WebCenter Sites* only if you have associated the *Tag Reference* and *Javadoc* with your current WebCenter Sites instance.

- **Sites Workspace Elements** provides access to code-related resources. This view shows the resources in a tree and groups each resource according to its site affiliation.

- **Logging Configuration** shows a dynamically updating view of the Log ODL configuration. In this view you can set the log levels of each WebCenter Sites logger.

- **Preview Browser** shows an embedded preview browser.

- **Template View**, shows all the Template assets created in Eclipse and any Template assets exported from WebCenter Sites to Eclipse.

See About Developer Tools Views.

28.4.3 How to Export and Import Data Between WebCenter Sites and Developer Tools

1. Start your local WebCenter Sites.

2. Start Eclipse.

3. Open the Oracle WebCenter Sites perspective:

   From the Eclipse menu, choose **Window**, then choose **Open Perspective**, then **Other**, and then **Oracle WebCenter Sites**.

4. Select the **SynchUp** icon. The synchronization tool enables you to either export data from WebCenter Sites to the Developer Tools workspace or import data to WebCenter Sites from your Developer Tools workspace.

   - For a quick overview of using the synchronization tool, see Data Synchronization (Export/Import) Tool.

   - For detailed information about synchronizing resources, see Synchronizing and Exchanging Data Using Developer Tools.

28.5 Uninstalling Developer Tools

Uninstalling Developer Tools only takes a few clicks.

To uninstall Developer Tools:

1. Close the Oracle WebCenter Sites Perspective (along with any views associated with the Oracle WebCenter Sites perspective).

2. From the **Eclipse** menu, choose **Window**, then choose **Preferences**.

   The Preferences dialog opens.

3. Select **Install/Update**, then click the **Uninstall or update** link.

Figure 28-14 Preferences: Install/Update
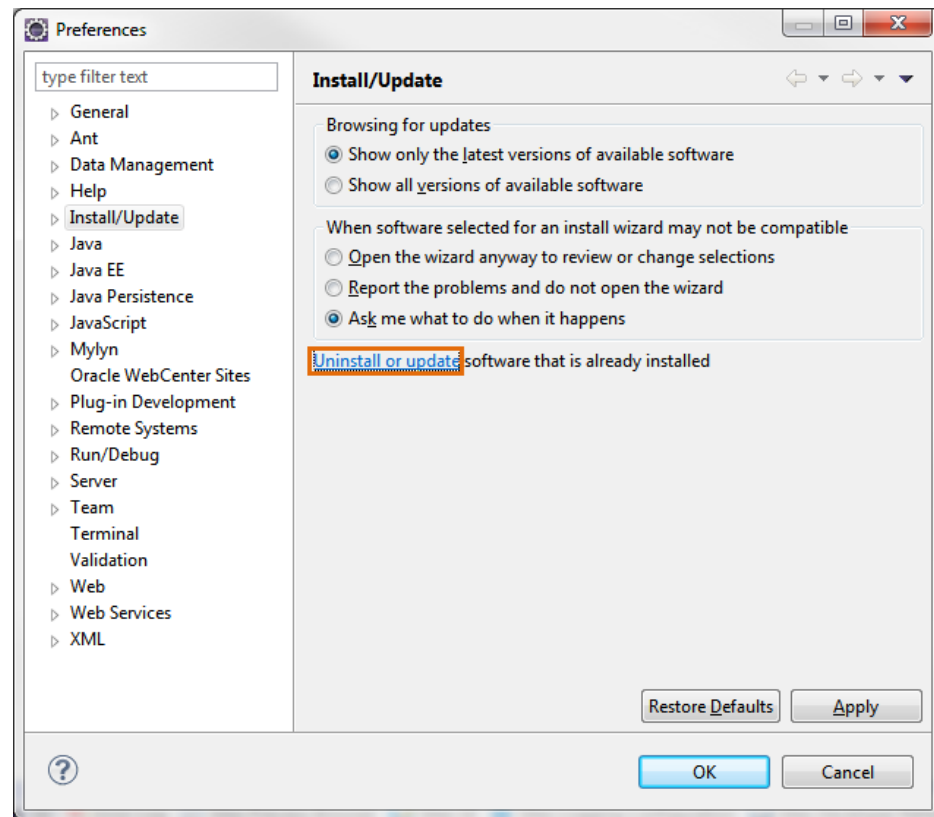
4. On the Eclipse Installation Details page, select the Developer Tools plug-in and then click **Uninstall** .

Figure 28-15   Eclipse Installation Details
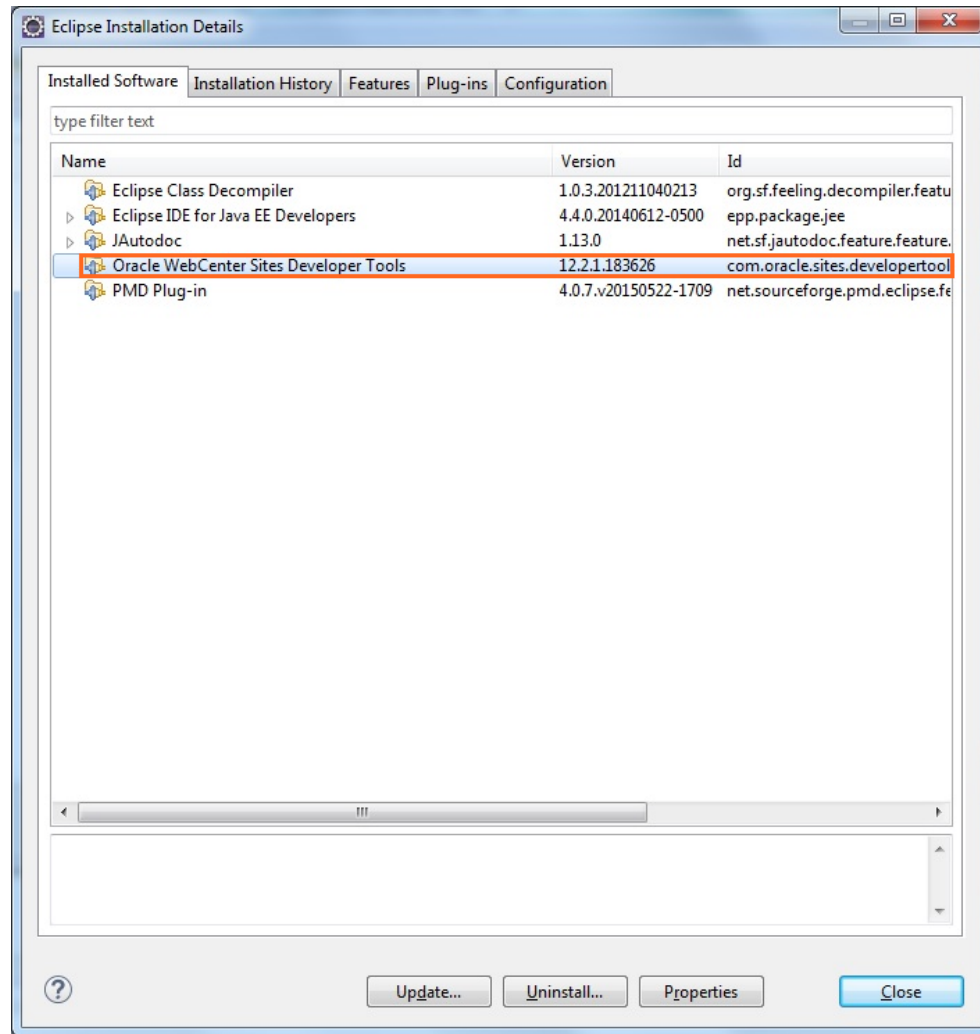
5. On the Uninstall Details page, click **Finish** to proceed with the process of uninstalling the Developer Tools plug-in.

6. After the update completes successfully, the Software Updates dialog opens. Click **Yes** to restart Eclipse.

   Restarting Eclipse refreshes the plug-in cache and completely removes the Developer Tools plug-in from Eclipse.

29

Introducing Developer Tools Features in Eclipse

In the WebCenter Sites-integrated Eclipse IDE, you can create projects, workspaces, and a variety of views, export and import data, and do much more.

Topics:

- About the Oracle WebCenter Sites Perspective

- Understanding the Configuration Form

- Understanding Projects and Workspaces in Eclipse

- About Developer Tools Views

- Data Synchronization (Export/Import) Tool

29.1 About the Oracle WebCenter Sites Perspective

All Developer Tools functionality in Eclipse is grouped under the Oracle WebCenter Sites perspective. However, you do not have to open the perspective to work with WebCenter Sites components. These components are available as long as Eclipse is integrated with a WebCenter Sites instance.

To open the Oracle WebCenter Sites perspective, select **Window**, then select **Open Perspective**, then **Other**, and then **Oracle WebCenter Sites**.

The following figure shows the Oracle WebCenter Sites perspective:

Figure 29-1   Oracle WebCenter Sites Perspective
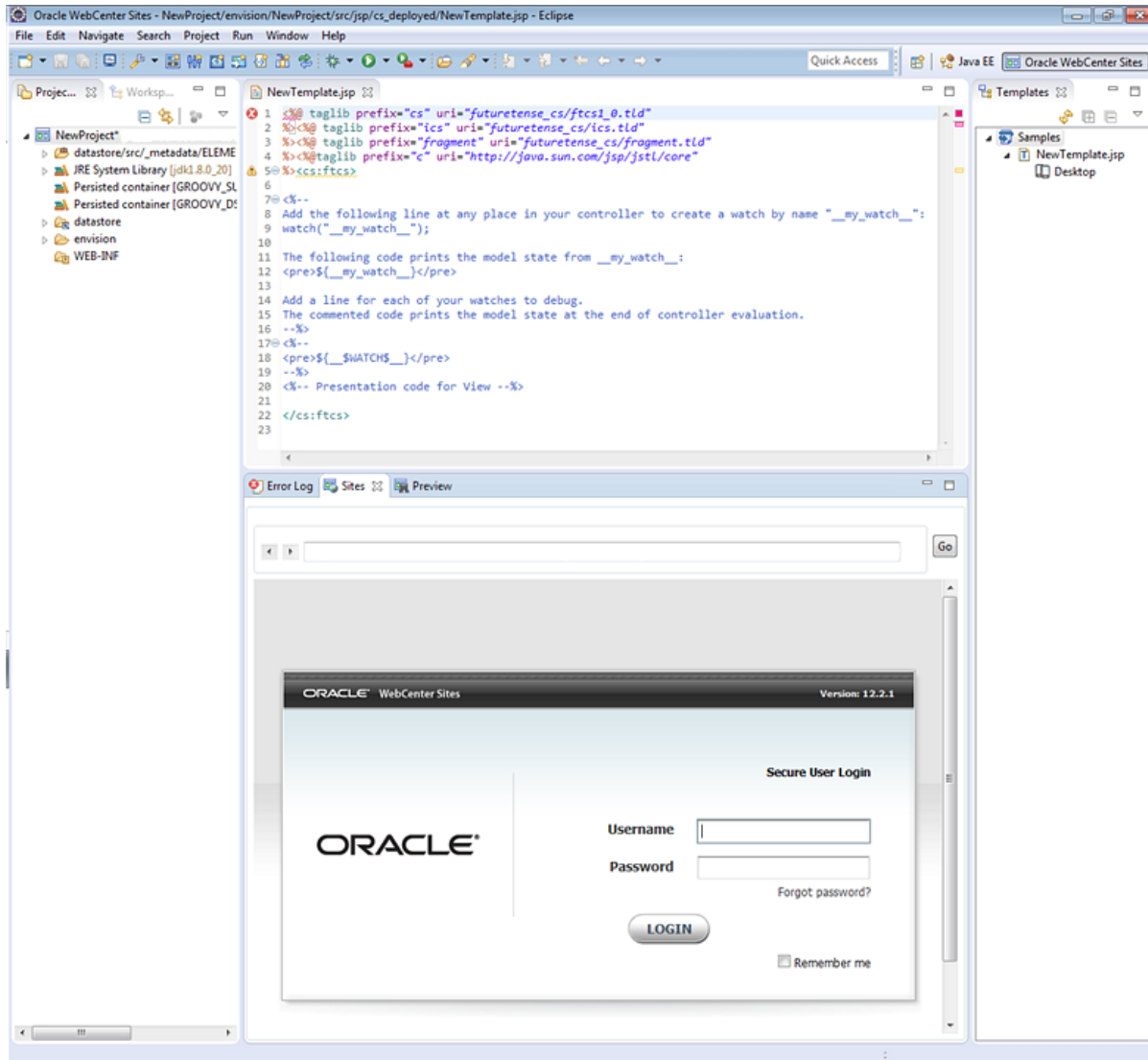
The following views are available in Eclipse when Developer Tools is successfully integrated with WebCenter Sites:

- Project Explorer and Sites Work (displayed in the left navigation pane)

- Templates View (displayed in the right navigation pane)

- Error Log, Sites UI, and Sites Preview Browser (displayed on the bottom of the Eclipse IDE)

- Wizards (displayed in the WebCenter Sites toolbar, at the top of the Eclipse IDE)

- Sites Log

- Sites Logging Configuration

- Sites Developer Reference

See About Developer Tools Viewsand Understanding Projects and Workspaces in Eclipse.

29.2 Understanding the Configuration Form

The configuration form opens automatically the first time you access a WebCenter Sites-integrated Eclipse. In this form you specify the WebCenter Sites instance with which you want to work.

On subsequent access, you can open the configuration form by selecting the **ConfigurePreference** wizard on the WebCenter Sites toolbar.

Figure 29-2   Configuration Form
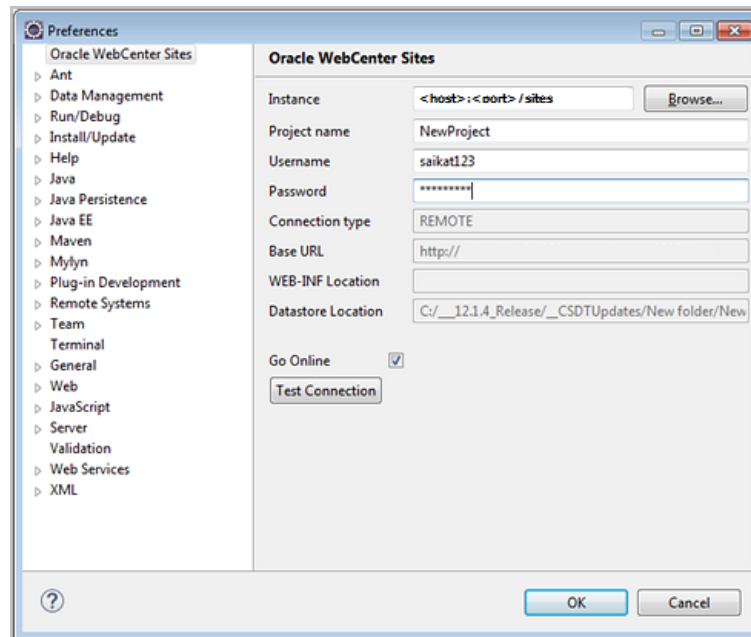
The configuration form requires the path to the WebCenter Sites installation directory, a WebCenter Sites user that is part of the `RestAdmin` group, and a project name for the WebCenter Sites instance. After you fill in all the required information, Developer Tools determines several other parameters for your WebCenter Sites instance, showing them in read-only fields. In addition, the **Test Connection** button enables you to determine whether Developer Tools is connected to the specified WebCenter Sites instance. The **Go Online** checkbox provides you the option of working online or offline. If you deselect this box, you will still have access to all the WebCenter Sites vies in Eclipse, however, you will not be able to create or edit WebCenter Sites resources. Click **OK** to create the project. If you chose to work online, clicking **OK** also logs you in to the WebCenter Sites instance and refreshes all the WebCenter Sites views.

29.3 Understanding Projects and Workspaces in Eclipse

Each WebCenter Sites instance that you access through Eclipse is assigned an Eclipse project. The Eclipse project's folder is displayed in the Project Explorer view. You'll need this project for tracking Developer Tools workspace items. Only one project is created by default for each WebCenter Sites instance and only one WebCenter Sites instance can be serviced by a project.

> **Note:**
>
> The main purpose of the project is to facilitate information tracking and process Eclipse events. Projects are managed by the Developer Tools plug-in. **Do not open, close, or modify the project.**

Each Eclipse project includes the following elements:

- `datasource`: This folder is linked to the location to which all your data is exported. For local connections, this folder is linked directly to the `config` folder under `<sites-home>` in your file system. For remote connections, this folder is created inside the project (envision) folder, which is linked to the datastore of the WebCenter Sites instance to which you are connected.

- `src`: By default, Developer Tools provides you with one `src` folder which contains all the Controller assets you have exported from WebCenter Sites. The resources in this folder can be checked in to a version control system. This folder remains empty until you export Controllers from WebCenter Sites to your Eclipse workspace.

- `WEB-INF`: This folder contains links to the current WebCenter Sites instance's `WEB-INF` folder. For remote connections, you must manually link this folder to the `WEB-INF` folder for the WebCenter Sites instance to which you are connected. See How to Enable Code Completion for Remote Hosts.

This figure shows a sample Eclipse project folder.

Figure 29-3   Sample Eclipse Project
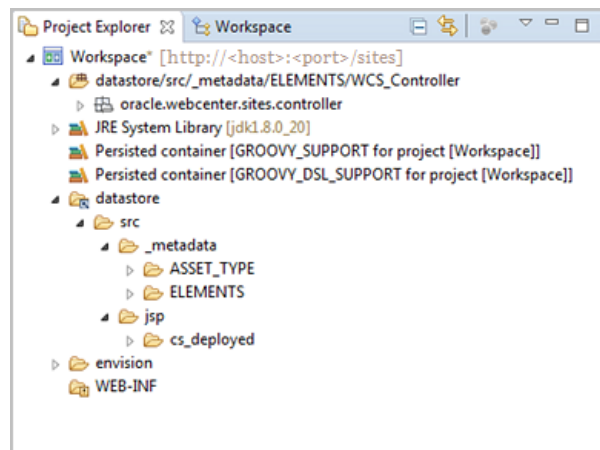
29.4 About Developer Tools Views

Several different views are available to help you preview different aspects of your site. For instance, you can preview code-related resources, assets, template assets in their hierarchy, pages, and even Admin and Contributor interfaces in an embedded browser.

See these topics about the views available in Developer Tools:

- Workspace

- Log Viewer

- Templates View

- Preview View

- Sites View

- Controllers View

- Logging Configuration View

- Developer Reference View

- Wizards

29.4.1 Workspace

This view provides access to code-related resources. The resources are grouped according to their site affiliation. When you select a resource, a quick summary of that resource is shown in the text box at the bottom of the view.

Figure 29-4   Workspace View
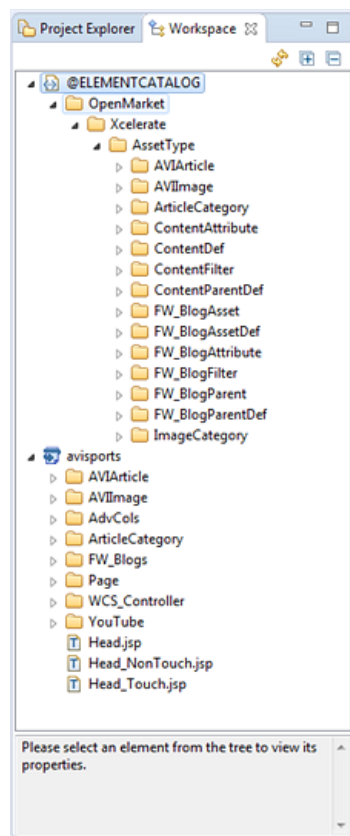
Right-click a resource in the tree to view the available management options. The options that are displayed depend on the resource you select:

- **Show Metadata**: Shortcut to the `.main.xml` file, which contains the metadata of the selected item.

- **Site Entry**: View the resource's site entry, share the site entry with other sites, create a new site entry, and delete a site entry.

- **Share**: Manage the sites with which this resource is associated.

- **Properties**: Manage properties of this resource, such as cache criteria and default arguments.

- **Delete**: Delete this resource.

29.4.2 Log Viewer

This view shows a dynamically updating record of the WebCenter Sites log file. This view can be used to monitor the behavior of your Eclipse-integrated WebCenter Sites instance.

Figure 29-5   Log File

29.4.3 Templates View

This view provides a hierarchical view of the Template assets you created in Eclipse and any Template assets you exported from WebCenter Sites into Eclipse. Click the **Sort** option to group templates by either their associated device groups or sites.

The following figure shows the Template view in the Eclipse IDE.

Figure 29-6   Templates View
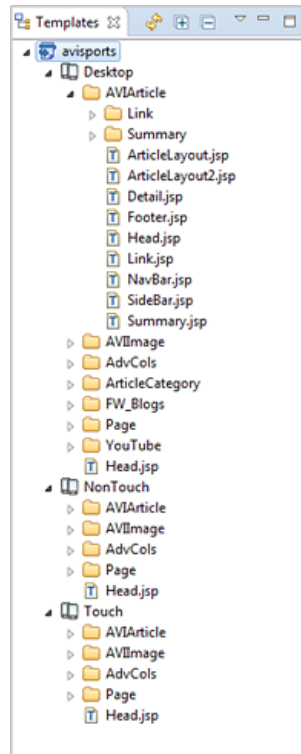
29.4.4 Preview View

This view provides a quick way to preview pages. To preview a web page with this view, enter the URL of the page in the address bar and press **Enter** or click **Go**. To refresh the current page, use the **Ctrl + R** keyboard shortcut or click **Go**.

Figure 29-7   Sites Preview Browser View
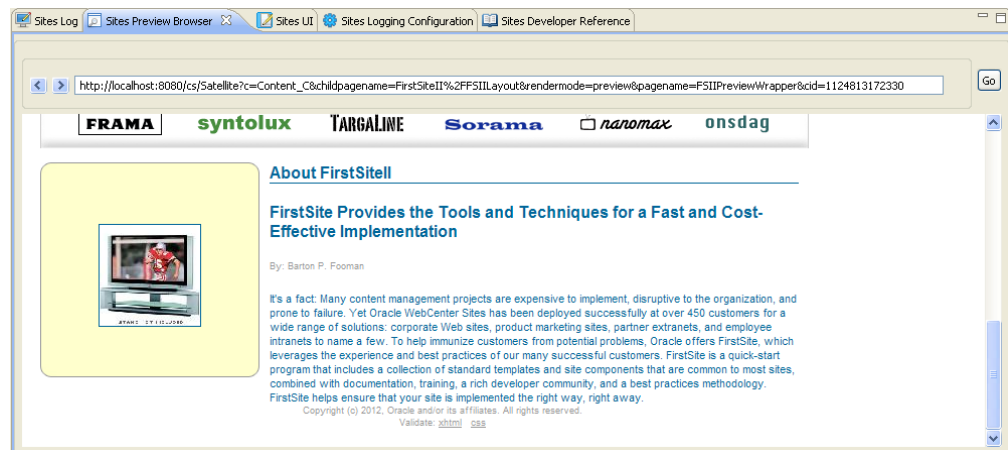
29.4.5 Sites View

This view shows the Admin and Oracle WebCenter Sites: Contributor interfaces in an embedded browser. This is equivalent to using either the Admin interface or Contributor interface in a standalone browser.

Figure 29-8   Sites View
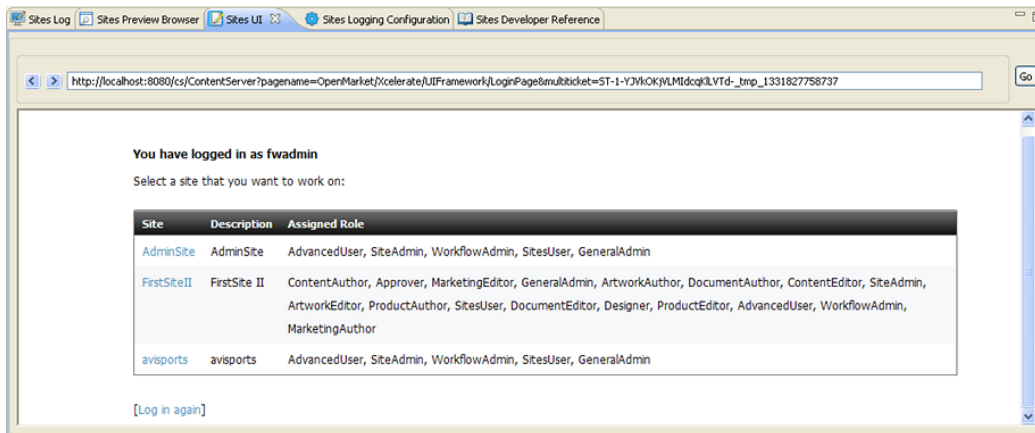
29.4.6 Controllers View

This view shows the Controller Asset Utility form, which is used to view and manage Controller assets.

This form is also available in the Admin interface (select the General Admin tree, expand Admin, then **System Tools**, and then double-click **Controller**).

Figure 29-9   Controllers View
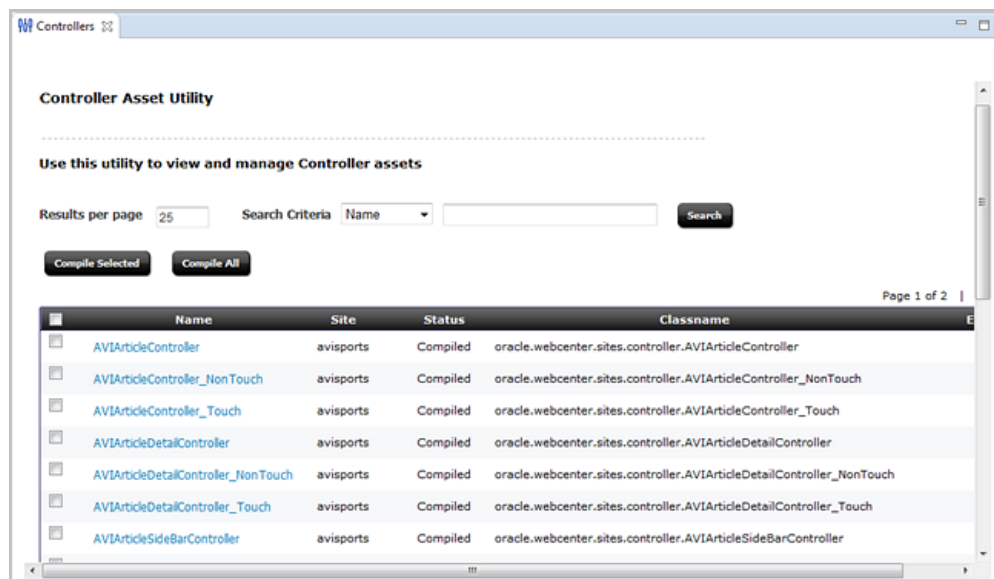
29.4.7 Logging Configuration View

If your WebCenter Sites system is using ODL, this view shows a dynamically updating Configure Log ODL form. The Configure Log ODL form enables you to view current loggers, change logger levels, add new loggers, and search logs.

Figure 29-10   Configure Log ODL Form

Num	Level	Logger	Set New Level
		Configure Log ODL	
		Configured Loggers	
1	NOTIFICATION:1	root	NOTIFICATION:1 TRACE:1 WARNING:1 ERROR:1 OFF
2	NOTIFICATION:1	oracle.wcsites	NOTIFICATION:1 TRACE:1 WARNING:1 ERROR:1 OFF
3	NOTIFICATION:1	oracle.wcsites.approval	NOTIFICATION:1 TRACE:1 WARNING:1 ERROR:1 OFF
4	NOTIFICATION:1	oracle.wcsites.auth	NOTIFICATION:1 TRACE:1 WARNING:1 ERROR:1 OFF
5	NOTIFICATION:1	oracle.wcsites.auth.request	NOTIFICATION:1 TRACE:1 WARNING:1 ERROR:1 OFF
6	NOTIFICATION:1	oracle.wcsites.blobserver	NOTIFICATION:1 TRACE:1 WARNING:1 ERROR:1 OFF
7	NOTIFICATION:1	oracle.wcsites.cache.page	NOTIFICATION:1 TRACE:1 WARNING:1 ERROR:1 OFF
8	NOTIFICATION:1	oracle.wcsites.cache.resultset	NOTIFICATION:1 TRACE:1 WARNING:1 ERROR:1 OFF
9	NOTIFICATION:1	oracle.wcsites.config.zookeeper	NOTIFICATION:1 TRACE:1 WARNING:1 ERROR:1 OFF
10	NOTIFICATION:1	oracle.wcsites.core.db.DBTransaction	NOTIFICATION:1 TRACE:1 WARNING:1 ERROR:1 OFF
11	NOTIFICATION:1	oracle.wcsites.core.http.HttpAccess	NOTIFICATION:1 TRACE:1 WARNING:1 ERROR:1 OFF
12	NOTIFICATION:1	oracle.wcsites.core.uri.definition	NOTIFICATION:1 TRACE:1 WARNING:1 ERROR:1 OFF

See Using the Configure Log ODL Tool in *Oracle Fusion Middleware Administering Oracle WebCenter Sites*.

29.4.8 Developer Reference View

This view contains tabs for *Oracle Fusion Middleware Tag Reference for Oracle WebCenter Sites Reference*, *Oracle Fusion Middleware Java API Reference for Oracle WebCenter Sites*, and *Oracle Fusion Middleware Java API Reference for Oracle WebCenter Sites: Visitor Services*.

29.4.9 Wizards

Wizards can be invoked from either the Oracle menu or the WebCenter Sites toolbar. They enable you to create code-based WebCenter Sites resources.

Figure 29-11   Wizards

The following wizards are available: **Create New Template**, **Create New Controller**, **Create New Element**, **Create New SiteEntry**, **Create New Element Catalog Entry**, and **Create New SiteCatalog Entry**.

See How to Create Resources.

29.5 Data Synchronization (Export/Import) Tool

This tool lets you synchronize resources in your workspace with those in your WebCenter Sites instance, and vice versa.

The data synchronization tool, accessible from the **SynchUp** wizard on the WebCenter Sites toolbar, provides you with two tabs:

- Export (Sync Resources to Workspace from WebCenter Sites)

29.5.1 Export (Sync Resources to Workspace from WebCenter Sites)

The **Export** tab is used to export data from the IDE-integrated WebCenter Sites to your Developer Tools workspace. In the process, Developer Tools serializes selected resources (transforms database representations into files) and copies the serialized representation to the Developer Tools workspace. You then can modify the resources in Eclipse.

The following figure shows the **Export** tab in the Synchronize Data form:

Figure 29-12   SynchUp Icon / Export Tab
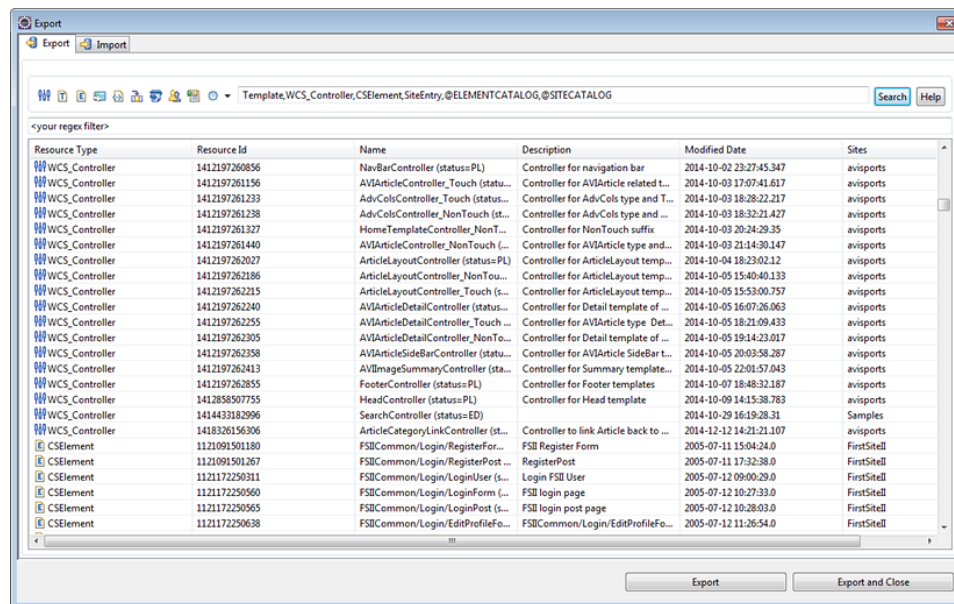
To export items from WebCenter Sites to your workspace:

1. Select the items you want to export. To narrow down the list of items, go to the **regex** search bar and enter the name of the asset type you are searching for. To search for multiple asset types, enter a comma-separated list.

2. Click **Export and Close**.

   The assets you exported to your Developer Tools workspace are now listed in the **Sites Work** tree tab.

29.5.2 Import (Sync Resources to WebCenter Sites from the Workspace)

The **Import** tab is used to import resources from your Developer Tools workspace into the IDE-integrated WebCenter Sites. In the process, Developer Tools transforms the selected resource to its native WebCenter Sites representation and copies it to the WebCenter Sites database.

The following figure shows the **Import** tab in the Synchronize Data form.

Figure 29-13   SynchUp Icon / Import Tab
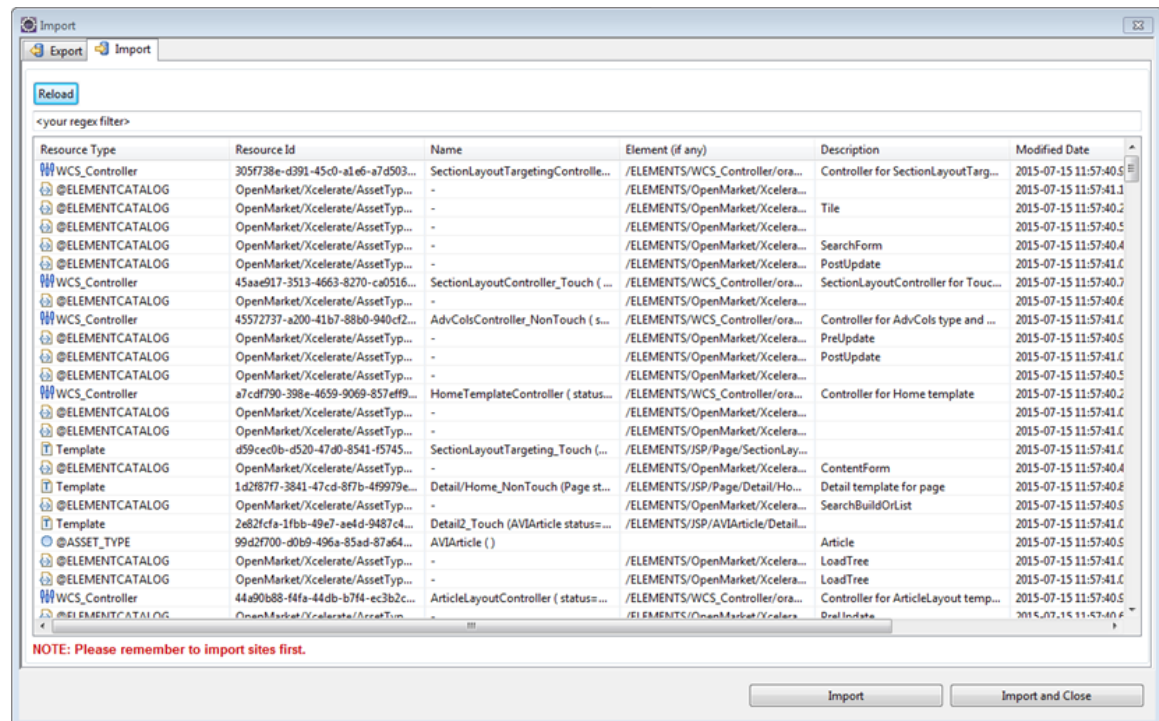
To import items into WebCenter Sites from the Workspace:

1. Select the items you want to import. To narrow down the list of items, go to the **regex** search bar and enter the name of the asset type you are searching for.

2. Click **Import and Close**.

30

Developing JSPs with Developer Tools

You can develop WebCenter Sites JSPs using the native Eclipse JSP editor using Developer Tools kit. Eclipse uses tag libraries and JAR files of your the current WebCenter Sites instance, and it also supports WebCenter Sites style of syntax highlighting and debugging.

Topics:

- JSP Development with Developer Tools

- Tag and Java API Completion

- Debugging

30.1 JSP Development with Developer Tools

Developing WebCenter Sites JSPs using the native Eclipse JSP editor is a convenient and productive experience. This is because the Eclipse JSP editor includes support for WebCenter Sites tag and Java API completion, syntax highlighting, and debugging.

The following figure shows an example of a WebCenter Sites JSP in the Eclipse editor:

Figure 30-1 Eclipse JSP Editor
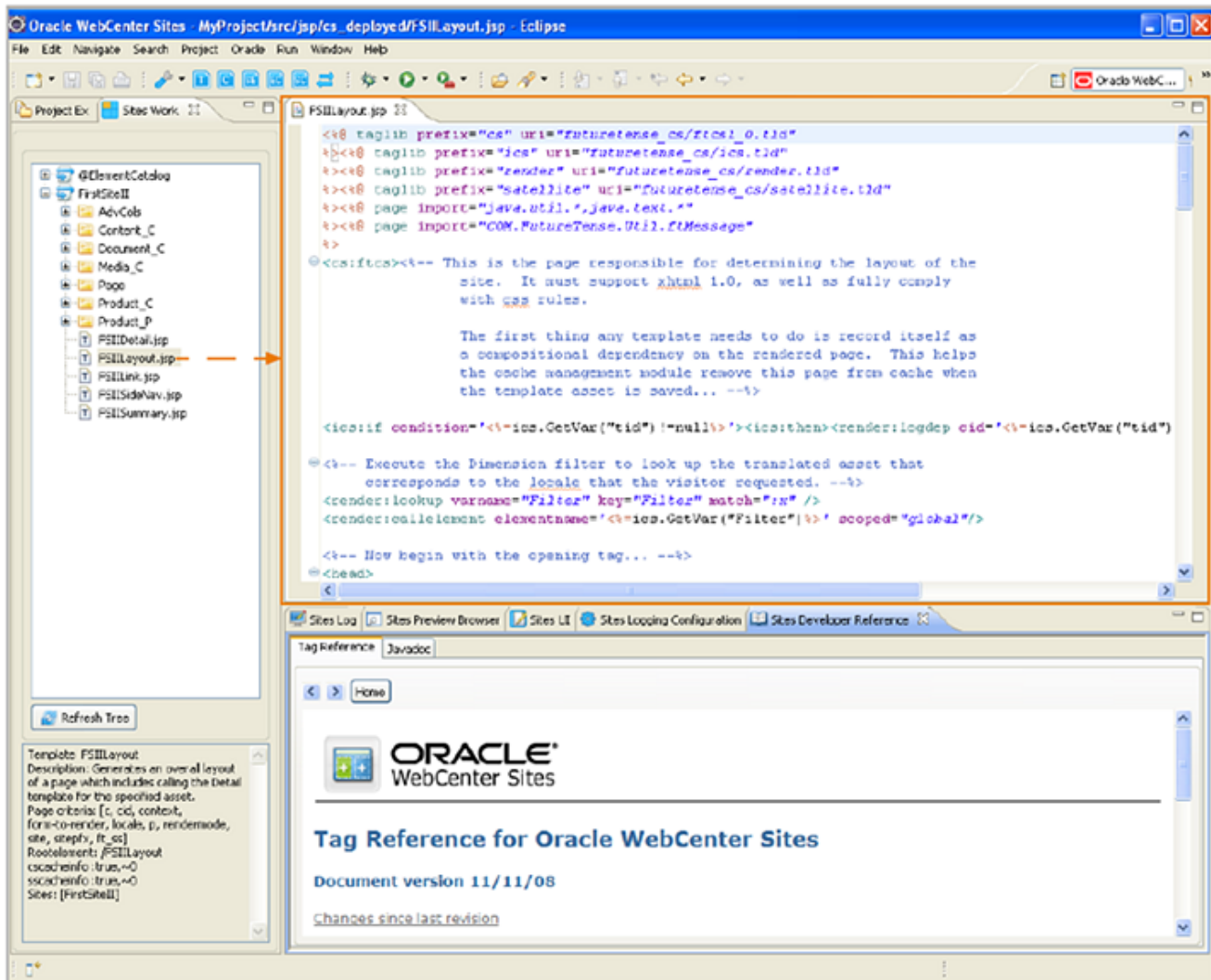
WebCenter Sites JSPs can include page caching, resultset caching, and associated metadata such as Template assets, CSElement assets, or ElementCatalog entries. The metadata of a JSP enables WebCenter Sites to track and manage it. Developer Tools handles a JSP's underlying WebCenter Sites processes transparently, including tracking the JSP and its corresponding metadata. If your WebCenter Site instance is running, and you save a JSP in Eclipse, the Developer Tools kit automatically synchronizes those changes with the WebCenter Sites instance. Any metadata associated with the JSP is also synchronized with WebCenter Sites. This enables you to view the changes in WebCenter Sites as soon as you save the JSP in Eclipse.

30.2 Tag and Java API Completion

In Eclipse you find tag and Java API completion features. Eclipse uses the tag libraries and JAR files of the current WebCenter Sites instance to provide the appropriate code completion for WebCenter Sites related tags and Java APIs.

For local hosts, the WebCenter Sites tag libraries and JAR files are automatically linked to your Eclipse project, and contained within the Eclipse project folder (located in the Project Explorer view). For remote hosts, the WebCenter Sites tag libraries and JAR files must be manually copied from the remote host to your Eclipse project. See Setting Up Developer Tools.

- The tag libraries are contained in the `futuretense_cs` folder under the `WEB-INF` folder.

- The JAR files are contained in the `WEB-INF/lib` folder.

Note:

When you use the tag and Java API completion feature, keep in mind the following:

- Make sure you follow strict JSP coding standards so that your code can be deployed on any application server.

- Eclipse code completion shows all public Java methods contained within the WebCenter Sites JARs. Only use the APIs described in the WebCenter Sites documentation. Using undocumented functionality is risky and unsupported.

In Eclipse, the tag and Java API completion features display information about each tag and piece of Java code you use when managing a WebCenter Sites JSP. For example, when you are working with a WebCenter Sites JSP and you begin to type the name of a tag, a window opens listing code completion suggestions. Then, a second dialog opens containing information about each suggestion.

Figure 30-2   Tag and Java API Completion Feature
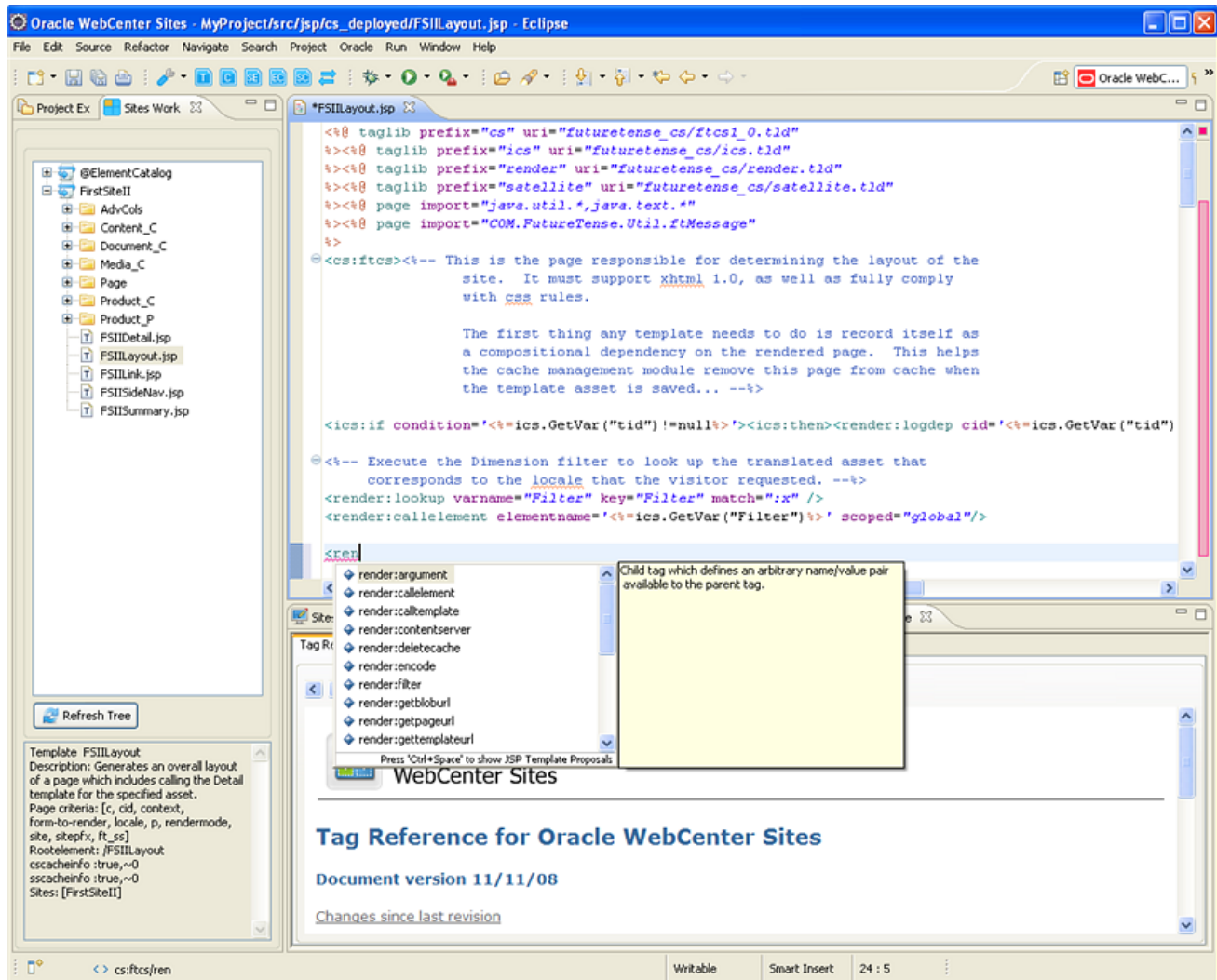
In addition to the code completion feature, the *Javadoc*, Visitor Services API, and *Tag Reference* are made accessible in the Sites Developer Reference view. See Sites Developer Reference View.

30.3 Debugging

If you would like to debug Java and JSP code in Developer Tools, you should first attach the debugger to the JVM process that runs WebCenter Sites. We recommend this especially when you plan to do remote debugging.

To attach the WebCenter Sites JVM, follow the instructions provided by Eclipse at the following URL:

http://www.ibm.com/developerworks/library/os-ecbug/

After the JVM is attached to the debugger, you can set breakpoints in your JSP and Java code, view variables, and so on.

31

Creating Templates for Mobile Websites Using Developer Tools

When you're creating templates for mobile websites, in the Sites view you can see templates under each asset type. And, the template variants that you create show up under each template. In the Device Groups (suffixes) view, you can view asset types under each device group instead. These views offer you a convenient development experience.

Topics:

- About Mobility Support in Developer Tools
- Creating Mobile Templates from the Sites Workspace Tab
- Creating Mobile Templates in Sites and Device Groups Views

31.1 About Mobility Support in Developer Tools

The Mobility feature in Developer Tools allows you to create websites for mobile devices.

See Configuring WebCenter Sites to Support Mobile Websites.

31.2 Creating Mobile Templates from the Sites Workspace Tab

The templates (default and mobile) you exported from your WebCenter Sites instance to Eclipse show under the Workspace tab. You can recognize mobile templates by their developer-defined suffix. For example, a template that you created for a touchscreen device may include the `_Touch` suffix.

You can create mobile templates from an existing template. To create a template for mobile websites:

1. In either the **Sites Workspace Elements** tab (located in the left panel), expand the node of the site for which you want to create a mobile template.

   > **Note:**
   >
   > A mobile template is associated with one or more device groups (that is, a group of devices with similar features) by a developer-defined suffix.

   This figure shows the expanded **MarketingSite** tree containing the `Home.jsp` template for the default device group (for desktop and laptop devices):

Figure 31-1   Home.jsp Template Element Displayed Under the MarketingSite Node
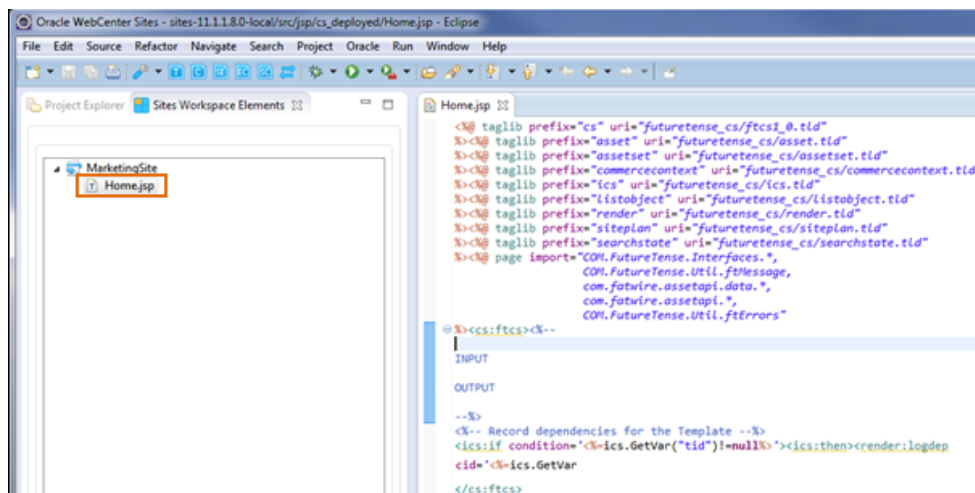
2. Right-click the template on which you want to base the mobile template. From the context menu, choose **Create Device Group Template** and then choose the name of the device group for which you want to create the template (see the next figure).

> **Note:**
>
> Multiple device groups can share the same suffix. If you do not see the device group in the list, a template defined by the same suffix as that device group may have been created. For more information about developer-defined suffixes, see Configuring WebCenter Sites to Support Mobile Websites.

Figure 31-2   Create Device Group Template Context Menu
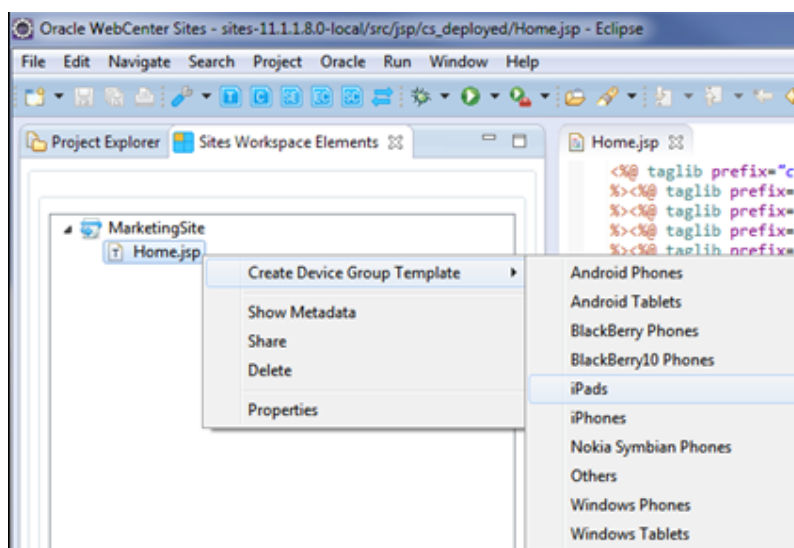
The Create New Template dialog opens showing fields with fixed and modifiable property values that the template wizard copied from the source template (the Home_Touch template of the **MarketingSite** in this example).

3. Edit the properties in the modifiable fields according to your mobile template requirements, and then click **Finish** .

Figure 31-3   Create New Template Form for the Home_Touch Mobile Template

The file name of the mobile template is listed in the **Sites Workspace Elements** tab.

Figure 31-4   Home_Touch.jsp Displayed in the Sites Workspace Elements Tab
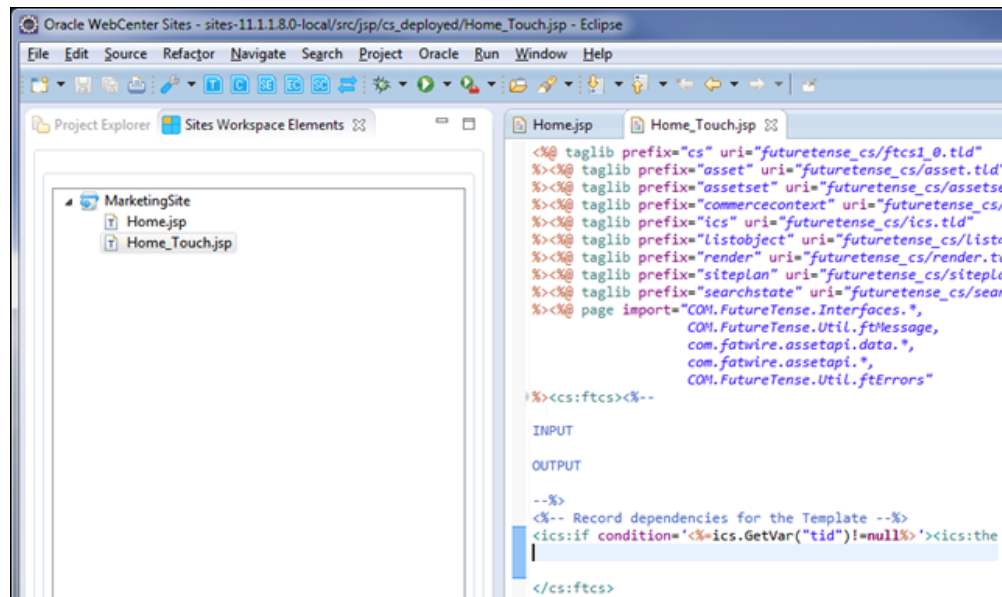
The mobile template is named **Home_Touch**, where **Home** is the name of the template and **_Touch** is the suffix defined for the device group for which this template was created.

4. Modify the code of your new mobile template in the native Eclipse JSP editor.

31.3 Creating Mobile Templates in Sites and Device Groups Views

You can view template variants by grouping templates by sites or suffixes. In the Sites view, templates are shown under each asset type and template variants under each template. This view is especially useful when you and other developers are working on different templates. This view lets you can create template variants anytime you want, without waiting for others to complete their work. In the Device Groups (suffixes) view, asset types are available under each device group so that developers of each suffix or device group can work simultaneously on their template variants.

To create template variants in Sites view:

1. On the toolbar, click the down arrow, then choose **Group by**, then **Sites** from the context menu.

Figure 31-5   Group by Sites
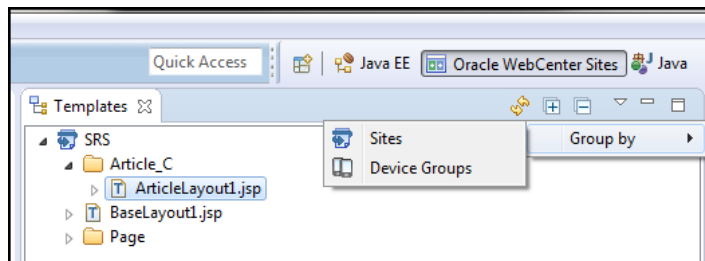
2. Under your site, expand template for which you wish to create variants. For example, if you wish to create a template variant for non-touch device group, you may choose the Touch template which may be quite similar to non-touch and create its variant. You can then edit this new variant for non-touch as required.

Figure 31-6   Site Tree Expanded to Show Suffixes
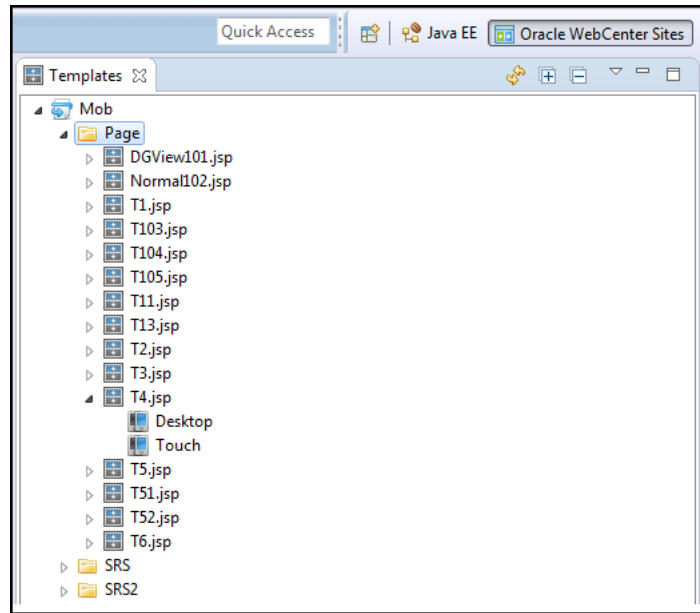
3. Do one of the following, as required:

    *   To create a template variant for a single device group, right-click a similar variant under the template, then choose **Create Device Group Template**, and then the device group for which you wish to create a template variant.

Figure 31-7   Create Device Group Template
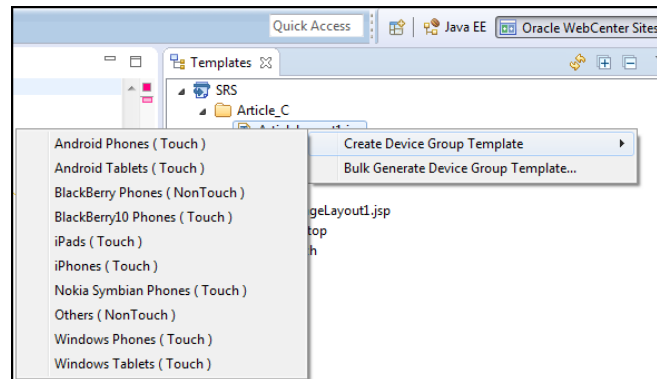
The Create New Template dialog pre-populated with corresponding values for the target variant is displayed.

Figure 31-8   Create a Single Template

Make your changes, if any, then click **Finish**. The new variant is displayed under the template for which you created it.

- To create template variants for multiple device groups, right-click the template, then choose **Bulk Generate Device Group Template**. In the multiple selection dialog, choose the device groups to create their template variants for the corresponding suffixes.

Figure 31-9   Multiple Variants Selection
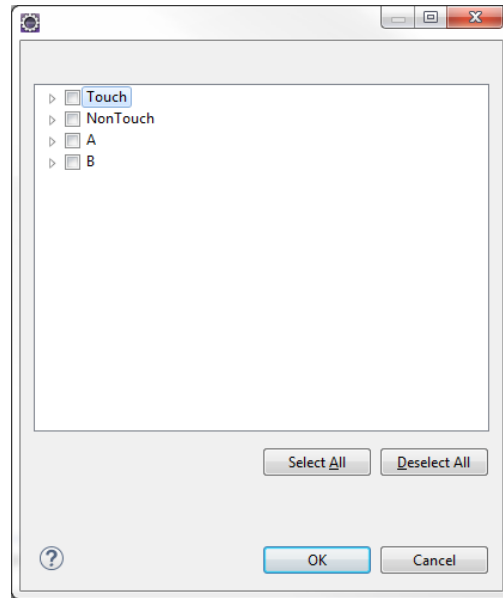
Click **OK**. The new variants are displayed under the template for which you created them.

4. Edit the template code as required, and save your changes.

To create template variants for device groups:

1. On the toolbar, click the down arrow, then choose **Group by**, then **Device Groups** from the context menu.

2. Expand the templates by device groups.

3. Under the asset type for which you wish to create a template, do one of the following, as required:

- To create a variant of a template, right-click the template under the asset type node, then choose **Create Device Group Template** , and then choose the device group for which you wish to create a template variant.

Figure 31-10   Template Variant Context Menu
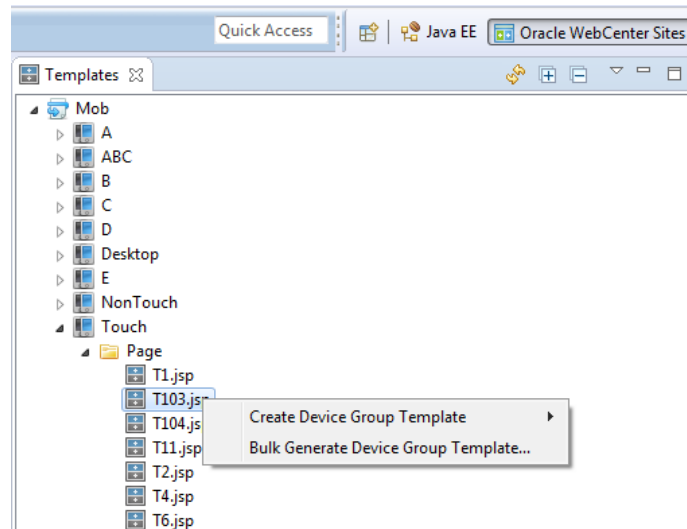
The Create New Template dialog pre-populated with corresponding values for the target variant is displayed.

- To create multiple variants of a template, right-click the template, choose **Bulk Generate Device Group Template**. In the multiple selection dialog, choose the device groups to create their template variants for the corresponding suffixes.

4. Edit the template code as required, and save your changes.

32

Synchronizing and Exchanging Data Using Developer Tools

The export/import feature lets you synchronize and exchange data. The Developer Tools kit uses ID and site mapping processes that enable you to exchange resources between WebCenter Sites instances.

Topics:

- Synchronization Using Developer Tools

- Synchronization Scenarios

- About Dependency Resolution

- ID Mapping

- Working with Site Mappings

32.1 Synchronization Using Developer Tools

Synchronization is the bidirectional flow of resources between a WebCenter Sites instance and its associated workspace. Using Developer Tools, you can export and import asset types, assets, site definitions, site catalogs, and also remap your site.

Using Developer Tools, you can perform these synchronization operations:

- Export/import assets with built-in dependency resolution and ID mapping

- Export/import asset types, such as flex families and AssetMaker asset types

- Export/import site definitions, roles, start menu items, and tree tabs

- Export/import SiteCatalog and ElementCatalog entries

- Perform site re-mapping; for example, creating reusable modules which can be imported into any WebCenter Sites CM site (command line interface operation)

Exporting or importing all resources of a given site enables you to track the entire site in a version control system. Advanced developers can use the command line interface to re-map the resources of one site to another by creating reusable modules (custom workspaces).

32.2 Synchronization Scenarios

Resources synchronize automatically, or you synchronize them as the need arises. Code-based resources synchronize automatically when you create or edit them with Developer Tools.

If WebCenter Sites is running, resources between WebCenter Sites and Eclipse are automatically synchronized when the following actions are performed in Eclipse:

- Code-based resources (Templates, CSElements, SiteEntries, ElementCatalog entries, and SiteCatalog entries) are created with the Developer Tools wizards in Eclipse.

- Code-based resources (Templates, CSElements, and ElementCatalog entries) stored in the Developer Tools workspace are edited in Eclipse. This includes edits to JSP files, XML files, metadata, and other files associated with the resource.

  For example, if you edit a resource's associated JSP file in the Eclipse editor, the Developer Tools kit automatically synchronizes the changes into the WebCenter Sites instance. Using the Eclipse editor, advanced developers can also edit metadata files (`.main.xml`) of flex definitions, and the Developer Tools kit automatically synchronizes the changes into WebCenter Sites. However, Oracle recommends using the Admin interface to modify flex definitions.

In certain cases, resources must be manually synchronized using either the Synchronization tool in the Eclipse IDE or (for advanced developers) the command line interface. Manual synchronization is required for the following situations:

- The Eclipse editor is not used to edit resources stored in the Developer Tools workspace; for example, when resources are copied to the Developer Tools workspace from a shared network file system or a version control system.

- WebCenter Sites resources are modified in the WebCenter Sites interfaces.

> **Note:**
>
> The Eclipse IDE provides an embedded Admin interface. However, Eclipse does not detect the changes that are made using this interface. Therefore, working in the embedded Admin interface is the same as working in a standalone browser running the Admin interface.

- WebCenter Sites is not running while you are creating or editing resources in the Eclipse IDE. After WebCenter Sites is restarted, you must manually synchronize the resources you created or edited.

The command line interface is used to synchronize resources mainly for deployment purposes, such as nightly builds that are deployed to test servers. For example, an advanced developer can embed a synchronization command into a script for an automated deployment procedure. See Using Developer Tools Command Line Interface (CLI).

32.3 About Dependency Resolution

WebCenter Sites resources often depend on other resources. For example, first you create a flex definition before you create the flex asset itself. In turn, the flex definition depends on a set of attributes and possibly other resources. Therefore, all flex constructs require that the flex family exist on the system.

To import a flex asset into an empty WebCenter Sites system, you must first create a flex family to which the flex asset will be associated. Then, create the following:

1. Create the flex attributes; for example, name, address, age, and so on.

2. Create the flex parent definitions.

3. Create flex definitions.

4. Create the flex parents.

5. Create flex assets.

When you export a flex asset, the Developer Tools kit performs all dependency resolutions for that asset and automatically exports all of its dependencies. Therefore, you only have to select the resource (such as the flex asset) and the Developer Tools kit computes all of the asset's dependencies.

> **Note:**
>
> The Developer Tools kit does not resolve a resource's dependency on site definitions. This enables you to choose whether you want to export or import an entire site, a subset of sites, or completely ignore site definitions (for example, if you are using the command line interface to create a reusable module that can be imported into any site). For a detailed example of creating a reusable module, see Using the Developer Tools Command Line Interface (CLI) to Create Reusable Modules.

32.4 ID Mapping

Each resource created in WebCenter Sites is assigned a unique local identifier. A resource's local identifier is unique to the WebCenter Sites instance on which it was created. Since multiple WebCenter Sites instances are used to create resources, it is possible for two different resources, on separate WebCenter Sites instances, to have the same local identifier.

See these topics:

- About ID Mapping

- Overriding a Resource's fw_uid

- What You Should Know About Using Developer Tools with Pre-Existing Resources

32.4.1 About ID Mapping

To uniquely identify resources, the Developer Tools kit assigns each resource a globally unique identifier (`fw_uid`), which is unique across all WebCenter Sites instances. In addition, when you import a resource into a WebCenter Sites instance, the Developer Tools kit assigns a new local identifier to that resource on that instance. If the resource references other assets (such as associations, asset pointers, and flex definitions), a new local identifier is generated for each of those assets. On subsequent imports to that WebCenter Sites instance, the resources are assigned the same local identifier. The Developer Tools kit maintains the resources' `fw_uid` values across all WebCenter Sites instances. If the resource and its referenced assets are imported back into their original WebCenter Sites instance, the Developer Tools kit re-maps their local identifiers back to their original value.

> **Note:**
>
> Certain WebCenter Sites resources, such as Template assets, flex attributes, and tree tabs have unique name constraints. To avoid name conflicts, make sure each resource is uniquely named across all WebCenter Sites instances.

For example, Developer A is working with a WebCenter Sites instance named CS1, and Developer B is working with a WebCenter Sites instance named CS2. Both developers created a completely different Template asset. Developer A created Template A, and Developer B created Template B. The two Template assets have different `fw_uid` values and different names. However, since local identifiers are randomly assigned, both Template assets, by chance, have been assigned the same local identifier (`12345`). Developers A and B want to exchange Template assets between each other's WebCenter Sites instances. Developer A wants to import Template B into the CS1 instance, and Developer B wants to import Template A into the CS2 instance.

The next figure illustrates the steps both developers take to exchange Template assets between their WebCenter Sites instances. Both Template assets' local identifiers are re-mapped when imported into the other developer's WebCenter Sites instance. When Template A is imported into the CS2 instance, the system assigns it the local identifier `52563`. When Template B is imported into the CS1 instance, the system assigns it the local identifier `22342`. In each case, the `fw_uid` values for both Template assets remain the same.

> **Note:**
>
> To exchange resources between WebCenter Sites instances, the developers in our examples use a VCS or shared file system. See Integrating Developer Tools Workspaces with Version Control Systems.

Figure 32-1   Exchanging Two Different Assets with the Same Local Identifier Between Two WebCenter Sites Instances
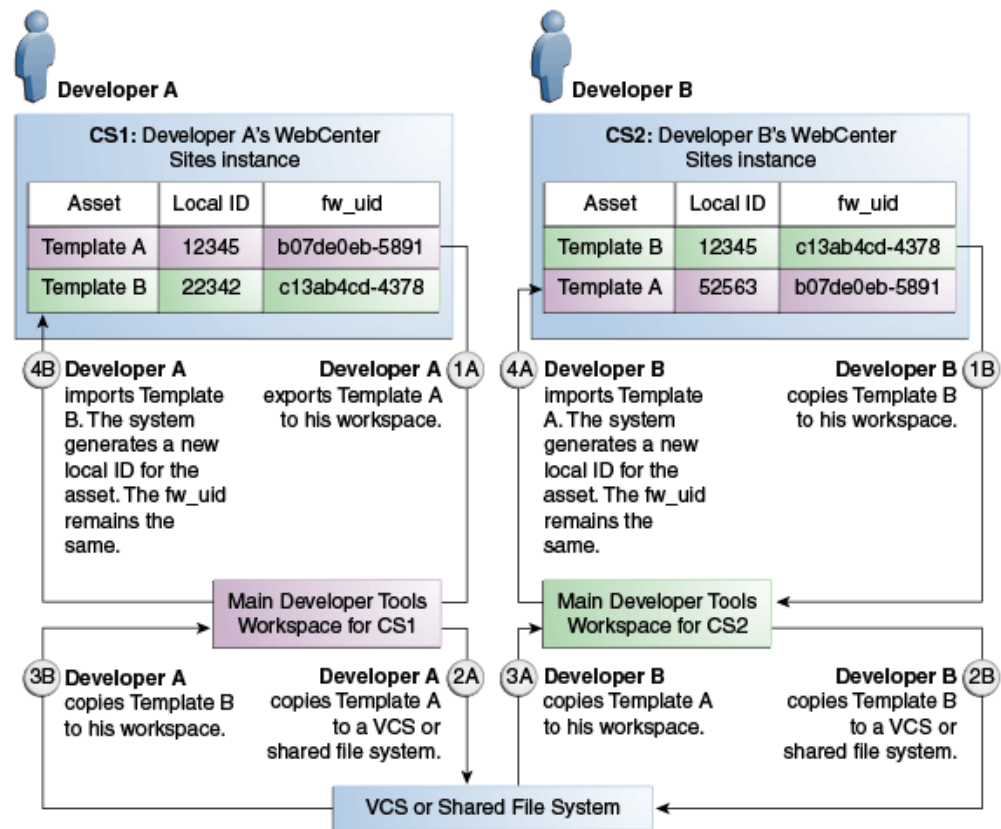
In the next figure, Developer A wants to deploy Template A to the Deployment WebCenter Sites instance (managed by the system administrator), and Developer B wants to deploy Template B to the same instance. Both Template assets have the same local identifier (12345).

Developers A and B each export their Template to the main Developer Tools workspace for their WebCenter Sites instance. They then copy their Templates to a VCS or shared file system. From here, the system administrator copies both Template assets to the Deployment WebCenter Sites' main Developer Tools workspace. The system administrator then imports the two Template assets from the workspace to the Deployment WebCenter Sites. Upon import, the system assigns both Templates a new local identifier. Template A is assigned the local identifier of 45678, and Template B is assigned the local identifier of 98765. The assets' fw_uid values remain the same.

Figure 32-2 Deploying Two Different Assets with the Same Local Identifier to a Third WebCenter Sites Instance
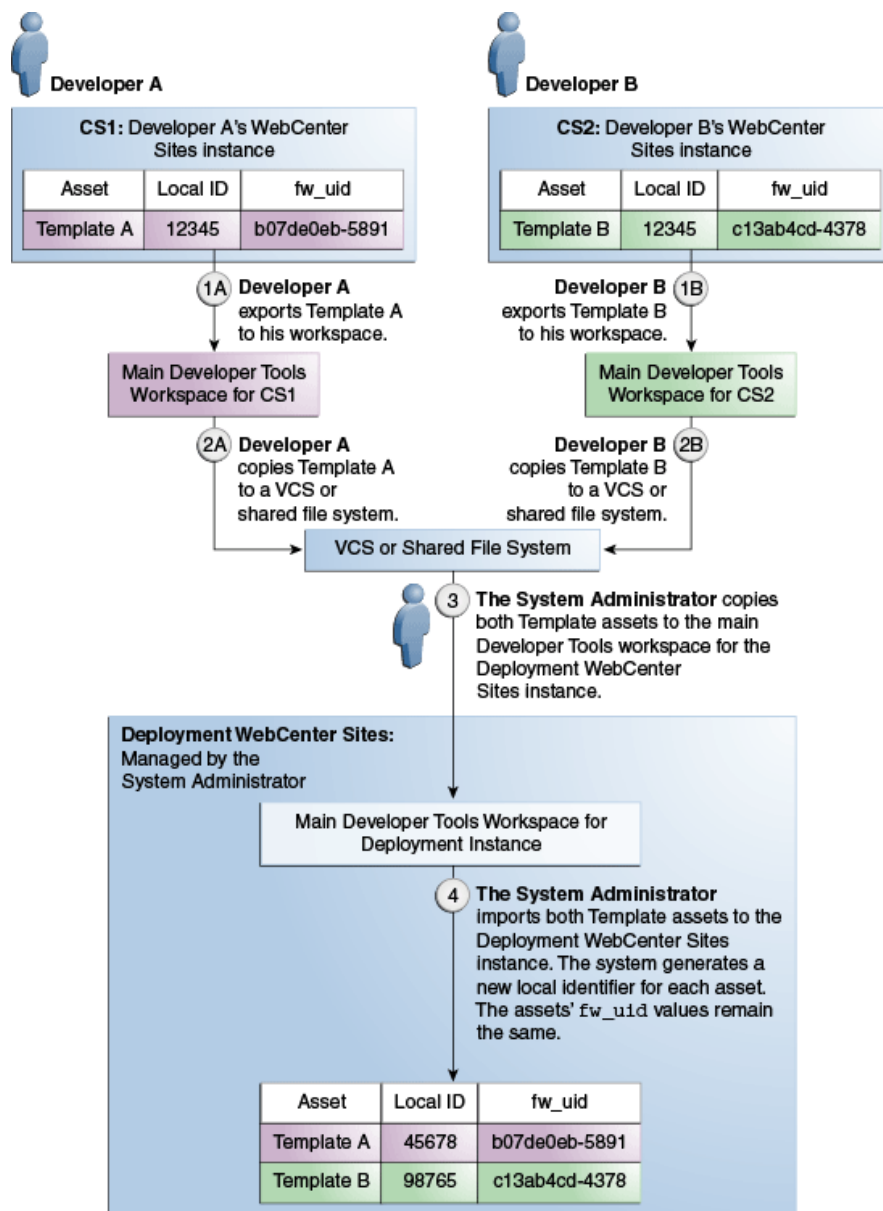
When a resource is exported to a workspace, it is identified by its `fw_uid`. ElementCatalog and SiteCatalog entries are not assigned an `fw_uid` because these entries are uniquely identified by element name.

32.4.2 Overriding a Resource's fw_uid

When a resource is created, a UUID value is automatically generated as its globally unique identifier and stored in an asset attribute named `fw_uid`. Advanced developers can use the Asset API to override the default `fw_uid` scheme with their own by modifying the `fw_uid` attribute. See the *Oracle Fusion Middleware Java API Reference for Oracle WebCenter Sites*.

> **Note:**
>
> Oracle recommends using the default WebCenter Sites `fw_uid` scheme. If you override a resource's default `fw_uid` value, make sure the value is unique across all WebCenter Sites instances. After you set a resource's `fw_uid` attribute, **do not** change the value.

32.4.3 What You Should Know About Using Developer Tools with Pre-Existing Resources

If your Oracle WebCenter Sites system is an upgrade from FatWire Content Server, some of its pre-existing resources may have their `fw_uid` values set to `CSSystem:[type]:id`. However, as of Content Server version 7.6, a resource's `fw_uid` is generated as a UUID value. Developer Tools can map resources with either type of `fw_uid` value, if the resource's `fw_uid` value is globally unique. Therefore, you can continue to use a pre-existing resource's current `fw_uid` value (in the format of `CSSystem:[type]:id`).

When using Developer Tools to work with pre-existing resources, do one of the following (or both):

- Oracle recommends continuing to use the pre-existing resource's `fw_uid` value of `CSSystem:[type]:id`. However, you must ensure that no other WebCenter Sites instance has generated the same `fw_uid` value for a different resource. For example, if you have a WebCenter Sites development instance and you published resources to a management instance, then the `fw_uid` values of the published resources remain the same on both instances. Therefore, synchronizing resources between these two instances using Developer Tools does not result in ID conflicts.

- If you have pre-existing resources that were created on separate FatWire Content Server instances but with identical `fw_uid` values, then each of those resources must be assigned a new, unique `fw_uid` value. To avoid ID conflicts, you can either remove the current `fw_uid` value and allow Developer Tools to generate a new UUID value when you export the resource from a WebCenter Sites instance, or you can assign your own unique identifier to the resource. See Overriding a Resource's fw_uid.

> **Note:**
>
> If you assign a resource a new `fw_uid`, make sure to assign the new `fw_uid` value to every instance of that resource. For example, if you published the resource to another WebCenter Sites instance before modifying its `fw_uid` value, make sure you assign the same `fw_uid` to both copies of that resource.

32.5 Working with Site Mappings

Most WebCenter Sites resources, such as assets, are associated with at least one site. When a resource is exported from a WebCenter Sites instance to a workspace, it stores a complete (canonical) list of sites with which it is associated in its `.main.xml` file. The resource's canonical list remains the same on every WebCenter Sites instance, unless you add a new site affiliation, remove a current one, or (if you are an advanced developer) override the resource's natural site mapping using the command line interface.

See these topics:

- About Natural Site Mappings

- About Overriding Natural Site Mappings With the Command Line Interface (CLI)

32.5.1 About Natural Site Mappings

By default, Developer Tools maps resources to their associated sites by referencing the canonical list stored in a resource's `.main.xml` file. If any of the sites referenced in this list exist on the WebCenter Sites instance to which the resource is imported, then Developer Tools maps the resource to those sites. If none of the sites referenced in the resource's canonical list exist on the WebCenter Sites instance, then the import fails.

For example, Developer A installs two sites: News and Sports. On a separate WebCenter Sites instance, Developer B also installs two sites: News and Weather. Both developers import the same Template asset into their WebCenter Sites instances. This Template asset is associated with both the Sports and Weather sites (both sites are referenced in the asset's canonical list). Upon import, Developer Tools references the Template asset's canonical list and then maps the asset to the Sports site on Developer A's environment and the Weather site on Developer B's environment.

When Developers A and B share the changes they made to the Template asset with each other, Developer Tools maps the asset to the appropriate sites on both WebCenter Sites instances. The canonical list enables Developer Tools to recognize the sites with which the Template asset is associated, even when the asset is exported into an instance where some of those sites are not installed.

32.5.2 About Overriding Natural Site Mappings With the Command Line Interface (CLI)

Advanced developers can use the command line interface to import a resource into sites that are not referenced in its canonical list. This interface enables you to create reusable modules, which are workspaces containing resources that can be imported into any site.

For example, a developer creates a blogging solution within the FirstSiteII sample site. This solution includes resources such as a flex family, assets, and Templates. The developer wants the resources to be imported into various sites, including sites that do not exist yet. Since he is an advanced developer, he uses the command line interface to export the resources to an empty workspace, and then archives the content of this

workspace (using a `.zip` or `.tar` format). Using the command line interface, other developers can then customize the site mappings of the resources contained in this module and manually specify the sites into which the module is imported.

See Using Developer Tools Command Line Interface (CLI)and Using the Developer Tools Command Line Interface (CLI) to Create Reusable Modules.

33

Using Workspaces in Developer Tools

Developer Tools stores resources exported from an integrated WebCenter Sites instance. The storage structures for each type of resource such as assets, code-based resources, attribute editor, and asset type is different. Continue reading to know how each storage structure is designed.

For information about using workspaces in Developer Tools to store resources, see these topics:

- Introduction to Workspaces

- Workspace Structure

- Asset Storage Structure

- Code-Based Resource Storage Structure

- Attribute Editor Storage Structure

- Asset Type Storage Structure

33.1 Introduction to Workspaces

A workspace is a disk-based repository of serialized WebCenter Sites data that represents resources from either the workspace's WebCenter Sites instance or another instance's workspace. Workspaces can store any type of WebCenter Sites resource including assets, flex families, sites, and so on. Each workspace is associated with one WebCenter Sites instance.

By default, Eclipse provides each WebCenter Sites instance with a main Developer Tools workspace (located in the Eclipse project folder) that is used for continuous development when working in the Eclipse IDE. Custom workspaces can be created by advanced developers using the Developer Tools command line interface. See Using Developer Tools Command Line Interface (CLI). Custom workspaces can be used for special projects, such as creating modules.

With the use of a version control system (such as Subversion) or a shared file system, resources stored on one workspace can be exchanged with other workspaces. Any resource exported from a WebCenter Sites instance into the associated workspace can be copied to another WebCenter Sites instance's workspace. This makes the resource available for import into the second workspace's associated WebCenter Sites instance. For more information about sharing resources between different workspaces, see Integrating Developer Tools Workspaces with Version Control Systems.

33.2 Workspace Structure

All workspaces have the same structure. The main Developer Tools workspace is the only visible workspace in the Eclipse project folder.

Workspaces are created under the `export/envision` folder inside the WebCenter Sites installation directory. The main Developer Tools workspace is located under the `export/envision/cs_workspace` folder.

Each resource contained in a workspace is stored as a single file or several interrelated files. The main file for each resource ends in `.main.xml` and contains resource-specific metadata. This main file also contains links to other files associated with the resource (such as an attached document, a JSP file, or a blob). This enables each resource to be fully self-contained, if all of a resource's associated files are stored in the workspace. Otherwise, the resource is incomplete.

Multiple files of a resource are listed in the bottom section of the `.main.xml` file as `storable0`, `storable1`, and so on. The associated files of any given resource have similar names. This way, all of a resource's associated files appear together, except ElementCatalog entries which are stored separately to preserve their original root path.

The location of a resource's files in the workspace depends on the type of resource. The workspace is divided into the following sections:

- `src/_metadata`: The metadata section of a given resource which contains assets, asset types, sites, roles, and so on. In addition, legacy XML code is stored under the `ELEMENTS/` subfolder.

- `src/jsp/cs_deployed`: This section stores a resource's JSP file under its proper path.

Because workspaces have a highly consistent structure, resources from one workspace can be copied to another. As with all file system copy operations, ensure you are not overwriting files that have the same name.

33.3 Asset Storage Structure

Assets are stored under folders named `src/_metadata/ASSET/asset` type.

Under this structure, there is a two-level hash-based hierarchy, which contains asset data. The name of the asset file is based on the asset name and its `fw_uid` value. If the asset includes attached documents or blobs, then the file name is based on the asset name, attribute name, `fw_uid` value, and the name of the document or blob (if any).

For example, a `Document_C` asset named `FSII IES_Manual.pdf` contains an attached document called `IES_MDPlayer_Manual.pdf`. Therefore, this asset is stored as two separate files:

- The first is the `.main.xml` file, which contains the asset's metadata and links to the files associated with the asset:

```
.src/_metadata/ASSET/Document_C/8/0/FSII IES_MDPlayer_
    Manual.pdf(aa0b47b5-f558-49d4-a6ac2ee012d1b75).main.xml
```

- The second is the actual document, which is a PDF file in this example:

```
.src/_metadata/ASSET/Document_C/8/0/FSII IES_MDPlayer_
    Manual.pdf.FSIIDocumentFile(aa0b47b5-f558-49d4-8a6a-
    c2ee012d1b75).IES_MDPlayer_Manual.pdf
```

> **Note:**
>
> Because all file names of the asset are based on the asset's name, renaming the asset also renames the file. If you are tracking the asset in a VCS, then delete the file with the old name.

33.4 Code-Based Resource Storage Structure

Templates, CSElements, and ElementCatalog entries are stored under the storage path required by their code elements.

The JSP files associated with code-based resources are stored in the workspace under `src/jsp/cs_deployed`, and the XML elements are stored under `src/_metadata/ELEMENTS`. The metadata files of code-based resources are stored under the same name as the resource's JSP with the appended `.main.xml` extension. Therefore, the code-based resource's metadata, JSP, and XML files are grouped together in the workspace.

33.5 Attribute Editor Storage Structure

Attribute editors are tracked as assets, but also have implicit references to a set of ElementCatalog entries. An attribute editor's ElementCatalog entries are tracked independently.

For example, the TextArea editor uses the `OpenMarket/Gator/AttributeTypes/TEXTAREA` ElementCatalog entry, which is registered as a dependency. Developer Tools maintains the following files for the TextArea editor:

- The `.main.xml` file:

  ```
  src/_metadata/ASSET/AttrTypes/9/10/TextArea(e64f983d-9c7c-489baedb-
      476d56f8121e).main.xml
  ```

- The `urlxml` metadata file:

  ```
  src/_metadata/ASSET/AttrTypes/9/10/TextArea.urlxml(e64f983d-9c7c-
      489b-aedb-476d56f8121e).1095346398911.txt
  ```

- The ElementCatalog entry, tracked as an independent resource:

  – The `.main.xml` file of the ElementCatalog entry:

  ```
  src/_metadata/ELEMENTS/OpenMarket/Gator/AttributeTypes/TEXTAREA
        .xml.main.xml
  ```

  – The attribute editor's element code:

  ```
  src/_metadata/ELEMENTS/OpenMarket/Gator/AttributeTypes/
      TEXTAREA.xml
  ```

33.6 Asset Type Storage Structure

Asset types have a main metadata part and a set of elements.

For example, the following is the structure of a Page asset type:

- The main metadata of the page is stored in the `.main.xml` file:

  ```
  src/_metadata/Asset_Type/Page(b8d8ae9-14cc-4554-b80e-0c22e39a3ec8).main.xml
  ```

- The associated elements are tracked independently (each element has its own `.main.xml` file):

```
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    SearchForm.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    CheckDelete.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    ContentForm.xml.main.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    ContentDetails.xml.main.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    LoadSiteTree.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    IndexReplace.xml.main.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    LoadTree.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    IndexAdd.xml.main.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    SearchForm.xml.main.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    IndexReplace.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    PreviewPage.xml.main.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    LoadTree.xml.main.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    PreUpdate.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    Tile.xml.main.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    SimpleSearch.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    SimpleSearch.xml.main.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    ContentForm.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    AppendSelectDetailsSE.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    LoadSiteTree.xml.main.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    AppendSelectDetails.xml.main.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    ManageSchVars.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    PreviewPage.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    CheckDelete.xml.main.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    ManageSchVars.xml.main.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/
    Page/PreUpdate.xml.main.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    AppendSelectDetails.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    IndexCreateVerity.xml.main.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    ContentDetails.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    PostUpdate.xml
```

```
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    IndexAdd.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    IndexCreateVerity.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    Tile.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    AppendSelectDetailsSE.xml.main.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
    PostUpdate.xml.main.xml
```

34

Using Developer Tools Command Line Interface (CLI)

You use the Developer Tools CLI for deployment and other resource movement activities. CLI gives you freedom to work with any workspace, not just the Developer Tools workspace. The command line interface also include import and export features. For example, with CLI you can create reusable modules (workspaces). These workspaces include resources that you can import into any site.

For information about running and using CLI, see these topics:

- Running and Using the Command Line Interface (CLI)

- Example Commands

- About Importing Modules

- Status Codes for Operations Invoked from the Developer Tools CLI

34.1 Running and Using the Command Line Interface (CLI)

Running CLI involves a few easy steps.

To run the command line interface:

1. Unzip the `developer-tools-command-line-x.y.z.zip` file, which is located in the `clients` folder in your WebCenter Sites installation directory (`${Oracle Home}/wcsites/clients`).

2. Execute the `developer-tools-command-line-x.y.z.jar` file and include the Java EE libraries (for example, the Servlet API implementation) in the classpath as follows:

   ```
   java -Xbootclasspath/a:lib/servlet-api.jar -jar developer-tools-command-line.jar
   http://<hostname>:<port>/<context-path>/ContentServer username=<username>
   password=<password>
   cmd=export|import|listcs|listds [options]
   ```

   Replace the placeholder parameters with the information about your development environment and the command you want to run:

   - `http://<hostname>:<port>/<context-path>`: The URL of your local WebCenter Sites instance, including the `ContentServer` servlet (for example, `http://localhost:8080/cs/ContentServer`)

   - `username` and `password`: The user name and password of a WebCenter Sites general administrator. This user must be a member of the `RestAdmin` group (for example, `fwadmin/xceladmin`).

   - `cmd`: The command to execute. The following commands are available:

- export: Export data from WebCenter Sites to a workspace.

- import: Import data into WebCenter Sites from a workspace.

- listcs: List WebCenter Sites content.

- listds: List workspace content.

- options: Specify one of the following to either import or export:

- resources: Specify which resources you want to import or export in a semicolon-separated list of resource type and resource ID. To specify multiple resources, use a comma-separated list. To specify all resources of a given type, use the * symbol. To export a resource (to a workspace), specify the resource's local ID. For example, use resources=Content_C:12345;Product_C:* to export a specific Content_C asset and all Product_C assets.

    If you are importing a resource (to a WebCenter Sites instance), specify the resource's fw_uid. To get the resource's fw_uid, use the listds option.

    The following is a full listing of resource selectors:

    @SITE: Specify the sites.

    @ROLE: Specify the roles.

    @ASSET_TYPE: Specify the asset types.

    @TREETAB: Specify the tree tabs.

    @STARTMENU: Specify the start menu items.

    @ELEMENTCATALOG: Specify the ElementCatalog entries.

    @SITECATALOG: Specify the site catalog entries.

    @ALL_NONASSETS: Use this short-hand notation to select all non-asset resources.

    @ALL_ASSETS: Use this short-hand notation to select all available assets.

    asset type: Specify assets of a certain type.

Note:

To verify that selectors are picking up the correct resources before import or export, use listcs for export activities and listds for import activities. These commands fine-tune the selectors before execution by providing a list of the resources that will be moved.

Resources' dependencies are exported and imported automatically. However, dependencies are not listed using the listcs and listds commands.

User preferences such as bookmarks and saved searches are not imported/exported as part of the CSDT import/export.

- fromSites: Select resources from specified sites only.

- toSites: (Import only) Override the natural site affiliation during import with a comma-separated list of sites. Specified sites must exist on the target system.

    – `modifiedSince`: (Assets only) Select only resources that have been modified since the specified date. The date format is `yyyy-mm-dd hh:MM:ss`. The date is treated as UTC 0 time zone.

    – `datastore`: Specify the workspace you want to either export WebCenter Sites resources to or import WebCenter Sites resources from. If you do not specify a value for this parameter, the main Developer Tools workspace is specified by default. If you are exporting resources and specify a workspace that does not exist, CLI automatically creates the workspace and exports the resources to it.

34.2 Example Commands

Here is a list of sample export and import commands that you can run using the command line interface.

- This command exports the specified `Content_C` assets and all `Product_C` assets that belong to FirstSiteII and were modified since the specified date. Because no workspace is specified, the Developer Tools workspace is used by default:

```
java -Xbootclasspath/a:lib/servlet-api.jar -jar developer-tools-command-line-
x.y.z.jar http://localhost:8080/cs/ContentServer username=bob password=password
resources=Content_C:123432123423,11234234212,111234341234;Product_C:*
fromSites=FirstSiteII modifiedSince="2016-08-08 19:14:00" cmd=export
```

- This command imports the specified `Content_C` asset and all `Product_C` assets that are found in the workspace to the FirstSiteII site and were modified since the specified date. Because no workspace is specified, the Developer Tools workspace is used by default:

```
java -Xbootclasspath/a:lib/servlet-api.jar -jar developer-tools-command-line-
x.y.z.jar  http://localhost:8080/cs/ContentServer username=bob password=password
resources=Content_C:aad618e9-f04e-4ee4-b902-076224bb6f7b;Product_C:*
toSites=FirstSiteII modifiedSince="2016-09-17 12:09:00" cmd=import
```

- This command exports all resources from the site SecondSiteII into a workspace named `TheOutput`:

```
java -Xbootclasspath/a:lib/servlet-api.jar -jar developer-tools-command-line-
x.y.z.jar  http://localhost:8080/cs/ContentServer username=bob password=password
resources=@ALL_ASSETS:*;@ALL_NONASSETS:*  fromSites=SecondSiteII
datastore=TheOutput cmd=export
```

- This command imports all assets and tree tabs from the workspace named `TheInput` into the site `MySite`:

```
java -Xbootclasspath/a:lib/servlet-api.jar -jar developer-tools-command-line-
x.y.z.jar  http://localhost:8080/cs/ContentServer username=bob password=password
resources=@ALL_ASSETS:*;@TREETAB:* toSites=MySite  datastore=TheInput cmd=import
```

> **Note:** `modifiedSince` works only with `export|listcs|import`.

34.3 About Importing Modules

Modules are sets of related resources that you export from your WebCenter Sites instance into a given workspace. Modules are reusable, so you can import their content into any CM sites (even if the site is not listed in the resources' canonical list of sites).

The `datastore` parameter enables you to specify the workspace you want to either export WebCenter Sites resources to or import WebCenter Sites resources from. If you export WebCenter Sites resources to a workspace that does not exist, the command line interface automatically creates that workspace and exports the resources into it.

To import a module into a CM site, you must execute an import command. In the `datastore` parameter, specify the workspace that contains the resources and in the `toSites` parameter, specify the site(s) to which you want to import those resources. This imports the content of the workspace into the specified CM site(s).

34.4 Status Codes for Operations Invoked from the Developer Tools Command Line Interface (CLI)

The CLI for the Developer Tools return these status codes for each operation invoked.

Status Code	Description
SUCCESS -> 0	Represents a successful invocation of a developer tools command (for example, export / import). This status is returned only when the request has been processed without any errors either on the client or the server side and there is at least one resource which could be processed as a result of the invocation.
NO CONTENT -> 1	Represents a successful invocation of a developer tools command (for example, export / import). This status is returned only when the request has been processed without any errors either on the client or the server side, but there are no resources which could be processed as a result of the invocation.
IMPROPER SHUTDOWN -> 11	A rare error code due to an unexpected breakdown of the process invoked.
CONSOLE UNAVAILABLE -> 12	Client-side error code. It's returned when the CLI interacts using a system other than the system console.
RUNTIME SANITY CHECK FAILED -> 13	Client-side error code. It's returned when the runtime provided for running the application is not sufficient, for example, there are missing dependencies.
BAD REQUEST -> 40	Client side error code. It's returned when the usage is violated.
UNAUTHORIZED -> 41	Client side error code. It's returned when the credentials or roles are invalid or insufficient to perform the processing.
SERVICE ERROR -> 50	Server side error code. It's returned if an exception occurs while invoking the request.
RESPONSE PARSE ERROR -> 50	Server side error code. It's returned when the response sent from the server does not match the expectation of the client. This error code is currently the same as `SERVICE ERROR`, but maybe expanded in future.

35

Integrating Developer Tools Workspaces with Version Control Systems

The resources in the Developer Tools workspace are stored in a version control system (VCS). You can share these resources with other developers if the need arises.

Topics:

- About Version Control With Developer Tools

- About Integrating Developer Tools With a VCS

- Using a Developer Tools-Integrated VCS: Example

35.1 About Version Control With Developer Tools

With version control systems (VCS) you can create source code repositories. A VCS can provide advanced tools for versioning, branching, and managing source files.

The file system structure in which the Developer Tools workspace stores WebCenter Sites resources enables those resources to be stored on any VCS and enables complete CM sites to be tracked in a VCS.

35.2 About Integrating Developer Tools With a VCS

In case of some VCSs, you need to their plug-ins for checking in resources into the VCS directly from Eclipse. For example, the Eclipse IDE supports the Subclipse plug-in for the Subversion repository. This plug-in enables you to check resources into the Subversion directory directly from the Eclipse IDE.

The Developer Tools workspace is located in the `src` folder of the Eclipse project. This folder can be accessed directly from the WebCenter Sites installation directory (under `export/envision/cs_workspace/src`). To copy the content of your Developer Tools workspace folder to a VCS, you must first determine which VCS you want to use. Then, check-in the resources stored in the Developer Tools workspace to the VCS. The VCS you choose to use determines the steps you must take to check resources in from the Eclipse IDE.

The Developer Tools workspace stores all resources as one or more files, depending on the type of resource. If you check a resource into a VCS, you must also check-in all associated files of that resource. For example, an asset that contains attached documents (such as a PDF) is represented by a metadata file (`.main.xml`) and the associated document file(s). All associated files of the asset must be checked in to the VCS. Otherwise, the check-in fails. For a detailed description of the Developer Tools workspace layout and for information about how resources are mapped to workspace files, see Using Workspaces in Developer Tools.

> **Note:**
>
> Checking data into a VCS from the Developer Tools workspace does not require an extensive understanding of the Developer Tools workspace file structure. Instead, most VCS clients detect incremental changes to the Developer Tools workspace folder and indicate those changes during a VCS commit operation.

35.3 Using a Developer Tools-Integrated VCS: Example

You can exchange the resources–that you check WebCenter Sites into a VCS from your Developer Tools workspace–with other developers and also track changes to those resources over time.

The following is an example of a development team using a VCS to share WebCenter Sites resources:

Developer A creates a resource in WebCenter Sites and exports it to the Developer Tools workspace. Developer A then checks that resource into a VCS. From the VCS, Developer B then can check-out the resource to his own Developer Tools workspace. This developer now can modify the resource and then check the changes back into the VCS. Developer A, and the rest of the development team, now can see the changes made to the resource from the VCS. This enables the members of the development team to synchronize their Developer Tools workspaces with the most recent changes made to the resource. Additional developers can join the group by checking-out resources from the VCS into their own respective Developer Tools workspaces. As the project advances, the cycle of adding and modifying resources continues.

> **Note:**
>
> WebCenter Sites provides a revision tracking system for resources that are kept within a given WebCenter Sites instance. The WebCenter Sites revision tracking system cannot be integrated with a VCS.

36

Using Developer Tools to Manage and Exchange Resources

Consider this scenario: A team of developers uses Developer Tools to create a CM site and its resources. This team uses the synchronization tool provided by Developer Tools to manage and exchange resources between multiple WebCenter Sites instances. They deploy the CM site and its resources as a nightly build using the command line interface.

Topics:

- [Today: Develop a Site and Associated Resources](#)

- [Three Days Later... Deployment](#)

36.1 Today: Develop a Site and Associated Resources

These example tasks involve developing a site and associated resources.

7:14 am: The New Project is Assigned

Artie the architect wakes up and finds himself appointed the leader of a new web-based project.

7:34 am: Setting Up Developer Tools

Artie gets some coffee and installs a WebCenter Sites instance on his laptop. He then starts the Eclipse IDE and configures the Developer Tools kit.

> **Note:**
>
> To successfully integrate Eclipse with a WebCenter Sites instance, Artie must enter the user name and password of a general administrator. This user must be a member of the `RestAdmin` group.

Figure 36-1   Eclipse IDE
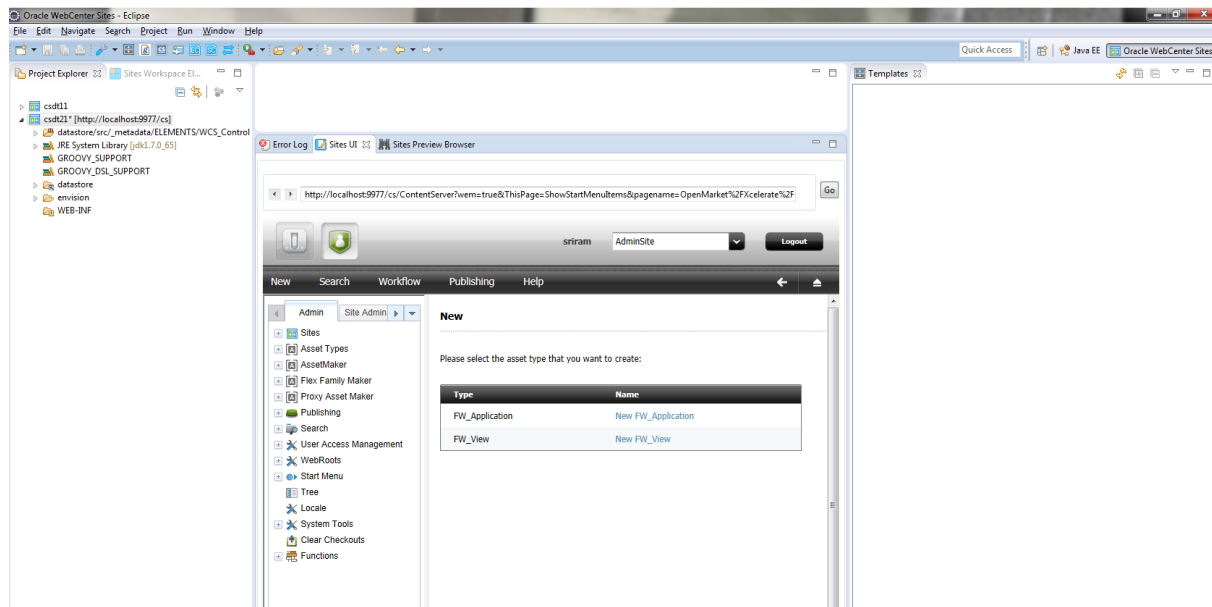
7:45 am: Create the Site Definition

Artie creates the site definition (naming the site `Acceptance`) by using the embedded Admin interface view in Eclipse.

Figure 36-2   Creating the Site Definition
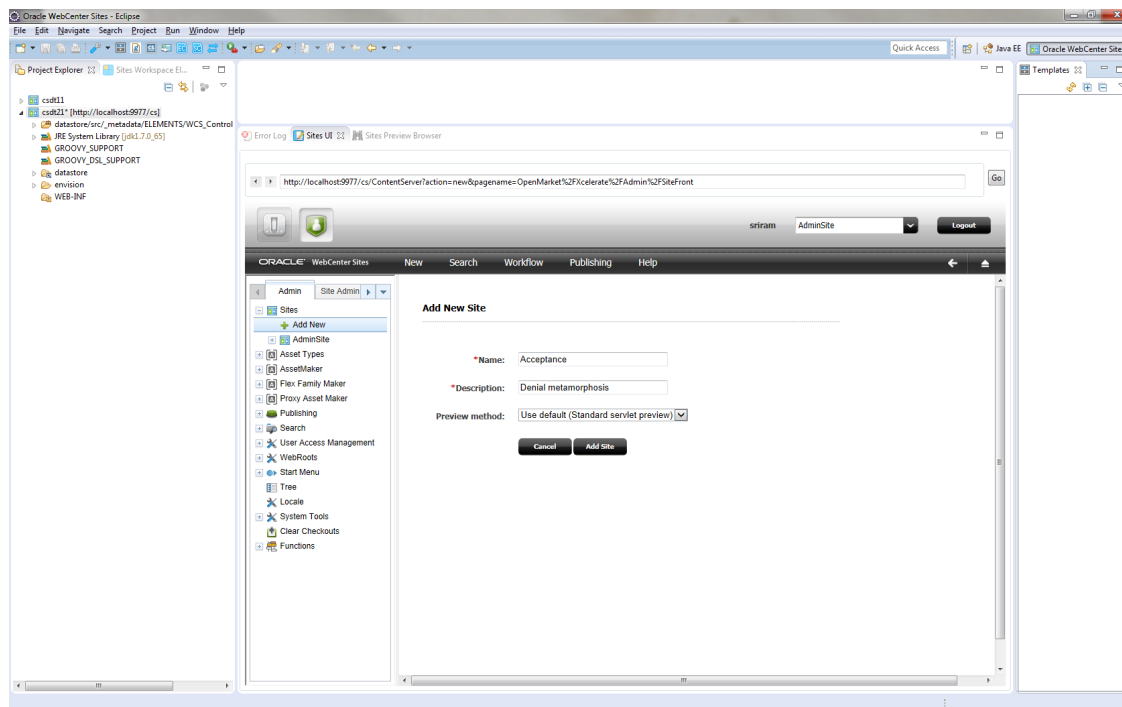
Artie could have used a separate browser window running the Admin interface to create the site definition. However, being a huge Eclipse fan, he indulges in the fact that he can usually write complete WebCenter Sites CM sites without leaving Eclipse.

7:46 am: Create Resources for the Site

Artie primes the site with the following resources:

- Enables asset types.

- Assigns permissions.

- Creates and enables a flex family to store information assets (author information assets in this scenario) for the site:

  - Flex Attribute: `Author_A`

  - Flex Parent Definition: `Author_PD`

  - Flex Definition: `Author_CD`

  - Flex Parent: `Author_P`

  - Flex Asset: `Author_C`

  - Flex Filter: `Author_F`

- Creates flex attributes (`authorName` and `authorBio`) and a flex definition (`fictionAuthor`). He then adds the attributes to the flex definition.

This figure shows this configuration:

Figure 36-3   Creating Resources for the Site
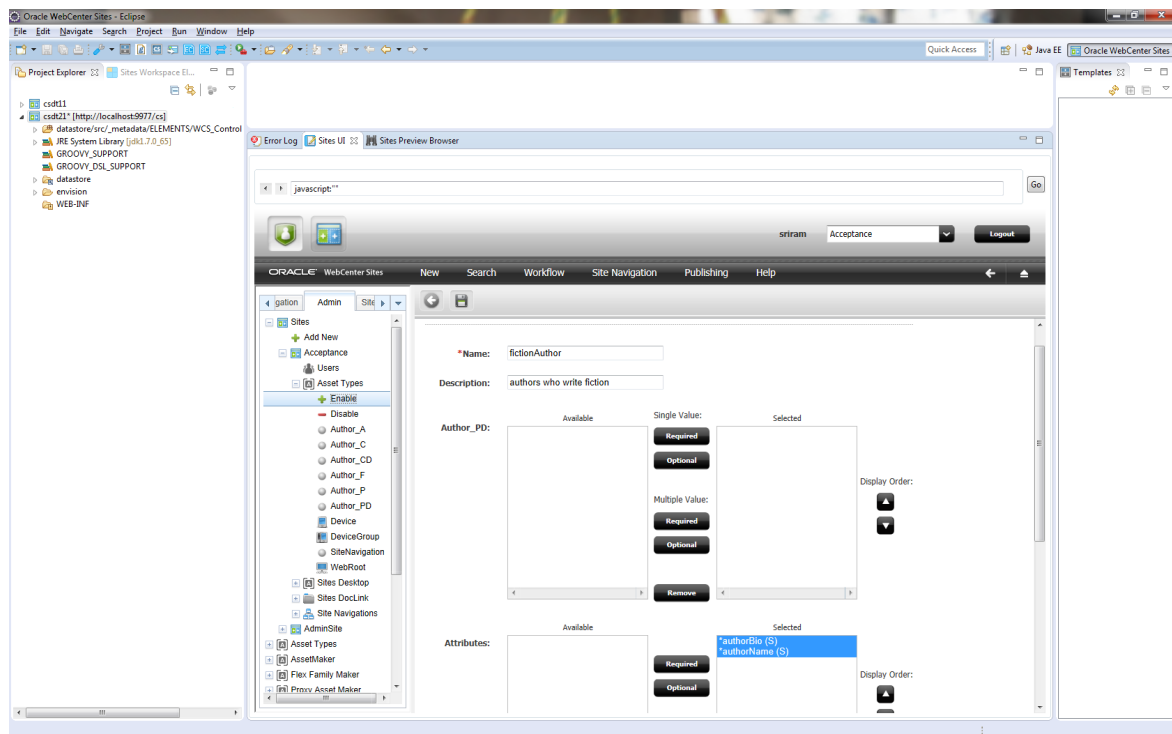
8:12 am: The VCS Discussion

Artie arrives at the office and meets with the rest of the development team: Sonoko (coder), Matthäus (coder), and Yogesh (system engineer). The discussion is about whether to use a version control system for the project:

Yogesh: I can set up a version control system in-house, but I would like to avoid doing extra work. Do you guys really want one?

Artie: Well, we expect this project to last several months. We could just create a shared folder on the network and synchronize all our work to it. However, we have to be careful not to overwrite each other's work. For example, if two people are working on the same Template asset, they have to wait for each other.

Sonoko: Artie, do you remember how the last project turned out to be very intense toward the end? Waiting for other people to finish their work is so unnerving when you have all this pressure from the management. I would much rather use a version control system. Also, can we keep the repository on the web this time so I can work from Stellarbucks when I'm bored?

Matthäus: I agree with Sonoko. We can get SVN hosting for next to nothing. We can even get an SVN with SSL for peace of mind.

Yogesh: If I don't have time to set up an in-house SVN, I could at least get you an SVN hosting subscription.

Artie: OK then, I guess we'll go with SVN. Anything else?

Artie and the rest of the development team decide to use SVN to track the resources of their site.

9:42 am: Synchronizing Workspaces With a VCS

Artie and his team install the Subclipse plug-in from http://subclipse.tigris.org/. Now, Artie needs to check-in the site and resources he created earlier:

1. Using Developer Tools Synchronization in Eclipse, Artie accesses the **Export** tab and enters the @Site selector in the **Search** field to retrieve a listing of all the sites on his WebCenter Sites instance.

Figure 36-4 Listing Sites on WebCenter Sites Instance
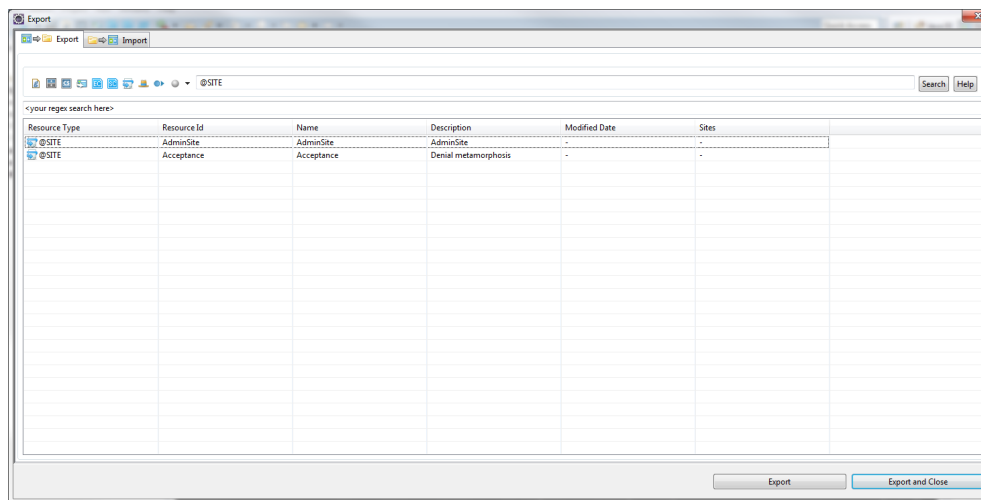
Artie selects the site he created earlier (Acceptance site) and clicks the **Export** button to export the site definition from his WebCenter Sites instance to his workspace.

2. Next, Artie exports the site's associated flex family to the workspace. He uses the @ASSET_TYPE selector to list all the assets on his WebCenter Sites instance. To narrow down the results, he uses the Author_ search string. Artie then selects all listed items and clicks **Export**.

Figure 36-5   Associating the Flex Family to the Workspace
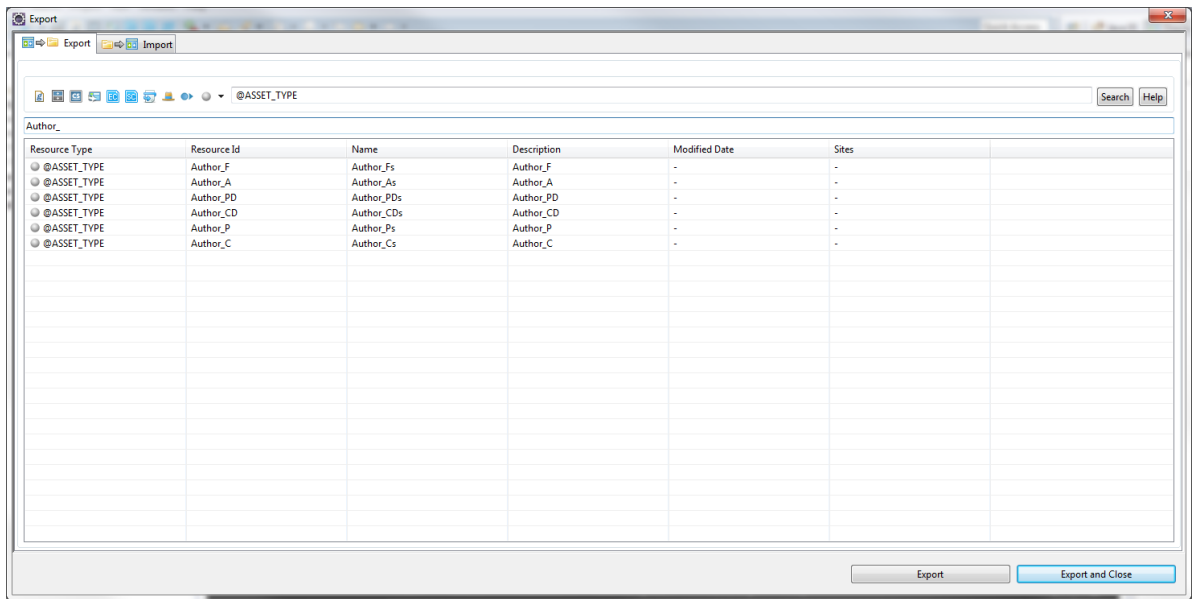
The flex family types are serialized to the workspace, including their type-specific ElementCatalog entries.

3. Now, Artie exports the flex definition to his workspace. He uses the `Author_CD` selector, which lists all available definitions of that type. In this case, there is only one definition (`fictionAuthor`).

Figure 36-6   Exporting the Flex Definition to the Workspace
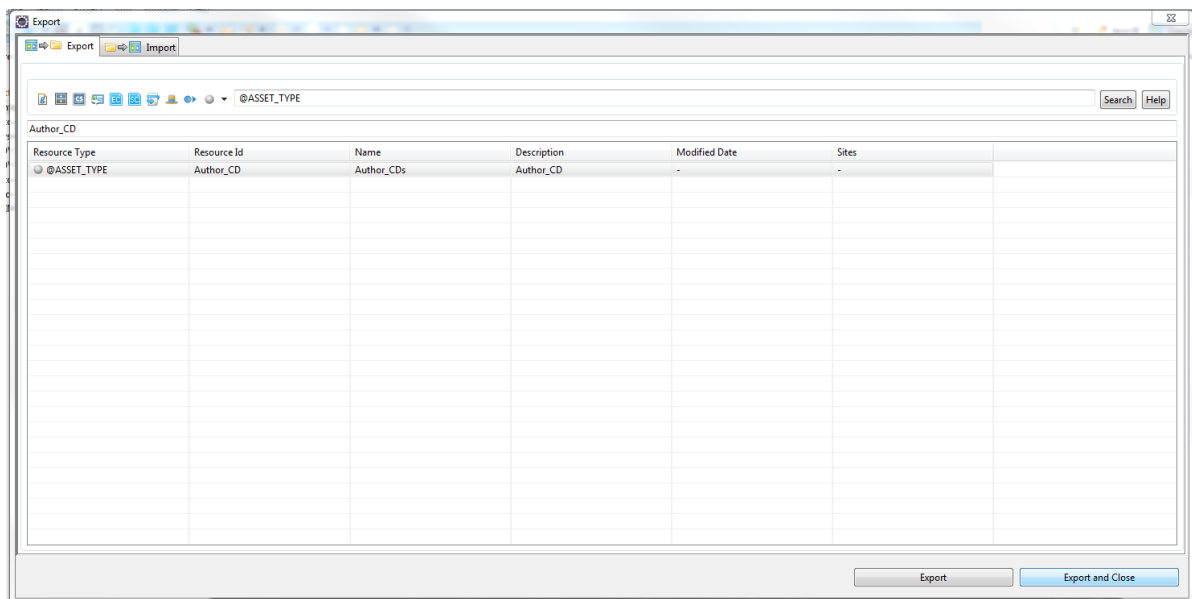
Note:

Artie did not select the flex attributes (`Author_A` instances) on which the site definition depends, because he knows the Developer Tools kit synchronizes them automatically with the definition.

4. Artie looks at his workspace in the Eclipse Project Explorer view to verify that all his work. From top to bottom, he sees the following under the project's `src` folder:

- `_metadata.ASSET_TYPE` entries for each asset type he synchronized.

- `_metadata.ASSET.Author_A` files for both of the Author_A attributes.

- `_metadata.ASSET.Author_CD` file for the serialized definition.

- `_metadata.ELEMENTS` entries for ElementCatalog entries related to each of the serialized asset types.

- `_metadata.SITE` entry for the site definition.

Figure 36-7   Workspace in the Project Explorer View
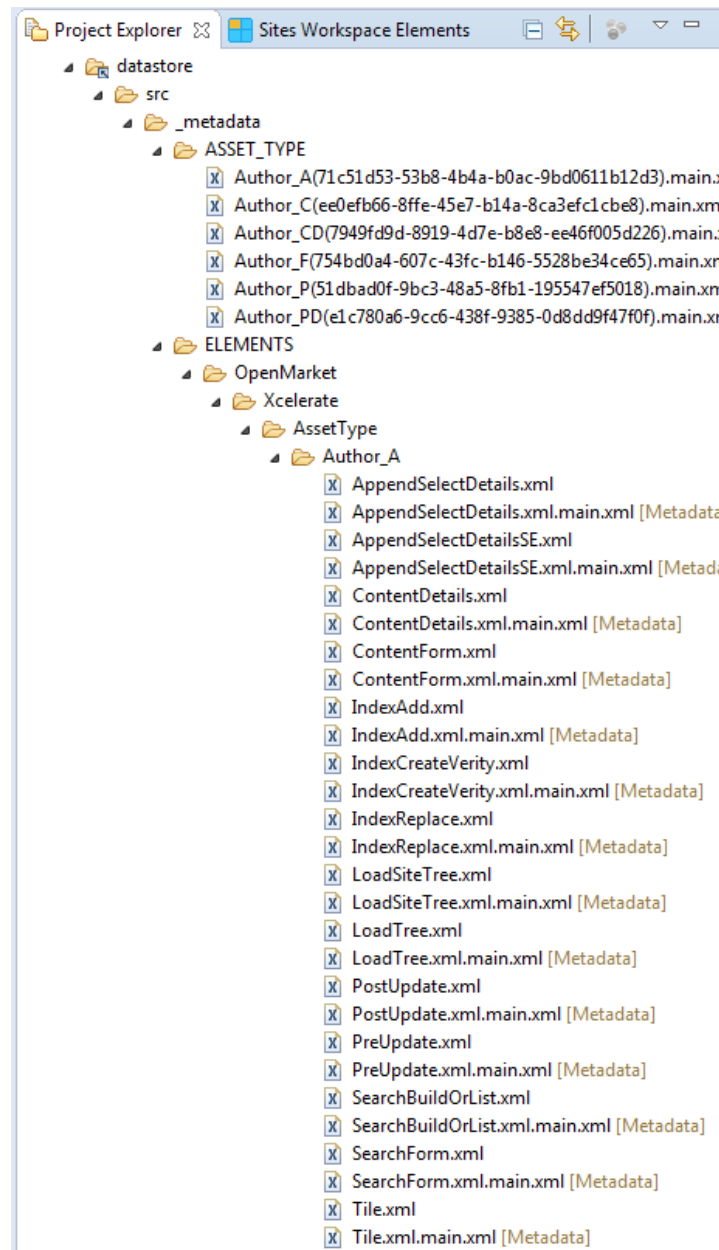
Note:

Artie could have looked in the `export/envision/cs_workspace` folder in his WebCenter Sites installation directory to see the same data.

Looks like all the resources are in Artie's workspace now. However, this is all on Artie's laptop and the team has no access to it. Time to check-in.

5. Using Subclipse, Artie connects to the development team's SVN repository and shares his Developer Tools project by committing his main Developer Tools workspace folder (`src` folder) to the SVN repository.

Figure 36-8   Committing the Project to SVN
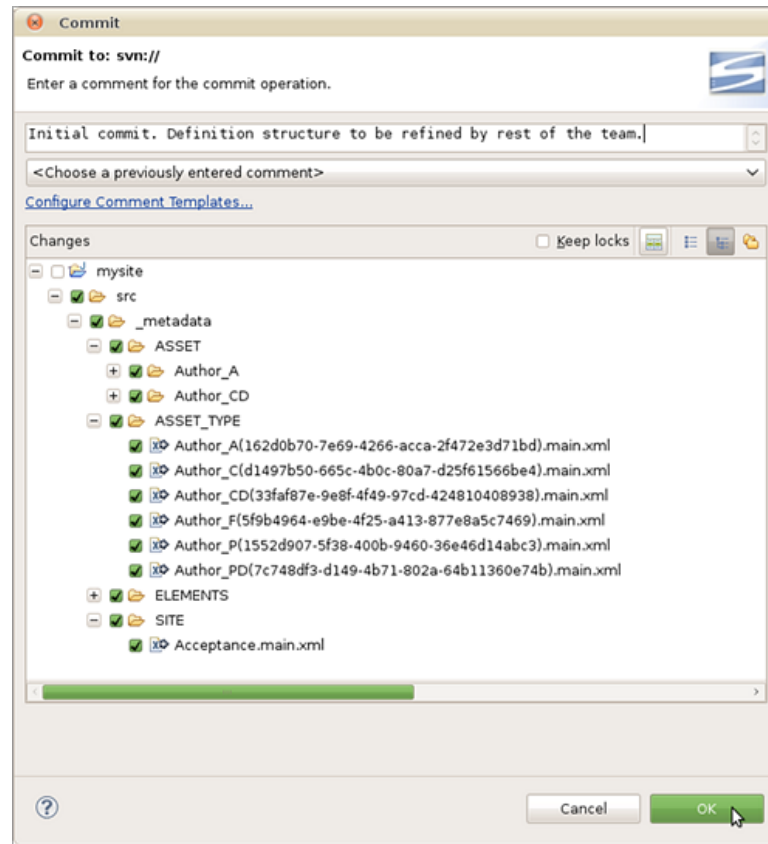
Note:

The main Developer Tools workspace is located under the `src` folder in the Eclipse Project Explorer view. Only commit the files that are located inside the `src` folder. All other files are auxiliary local resources and must not be committed.

10:12 am: The Other Team Members Synchronize their Workspaces to the SVN Repository

Sonoko and Matthäus just finished setting up their own, individual Eclipse-integrated WebCenter Sites instances. They both connect their Eclipse projects to the SVN repository.

Because Artie checked the site and its resources into the SVN repository earlier, Subclipse detects that the target location exists.

Figure 36-9   Remote Project Exists Dialog
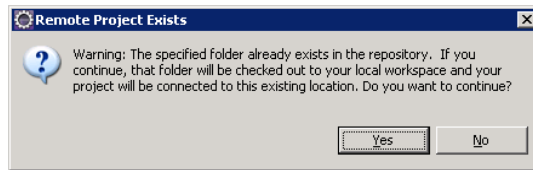
Sonoko and Matthäus both synchronize their main Developer Tools workspaces with the resources Artie made available in the SVN repository. Those resources are now accessible on both Sonoko's and Matthäus' main Developer Tools workspaces.

Figure 36-10   Synchronized Workspaces
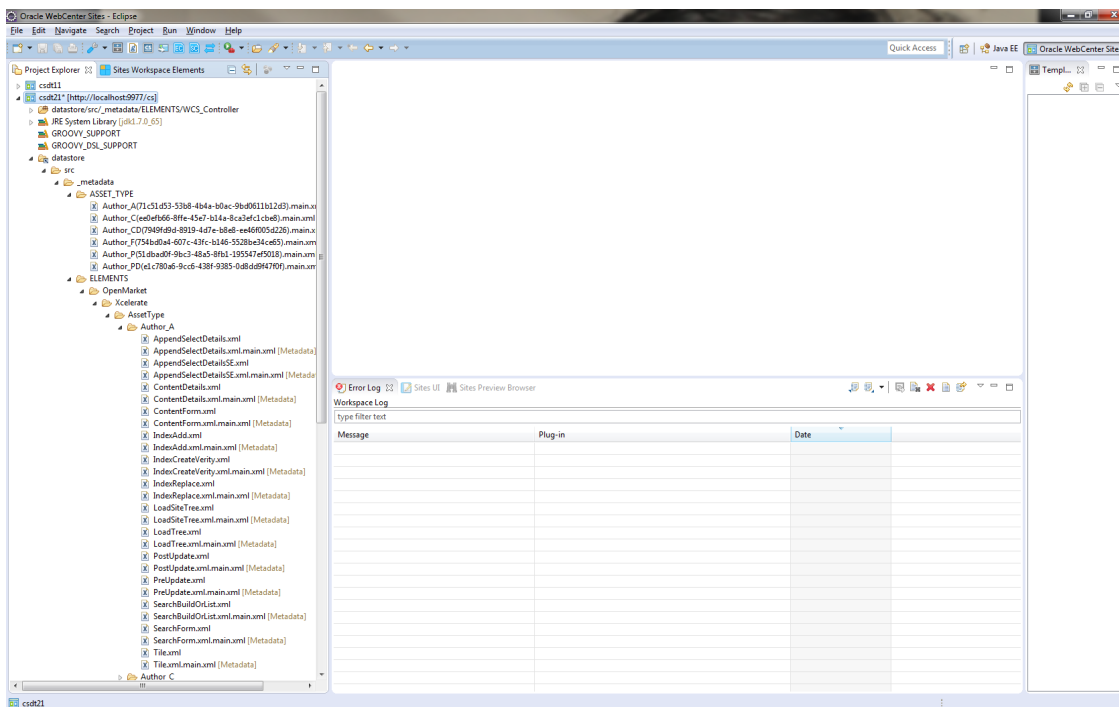
However, the resources are not synchronized with Sonoko's or Matthäus' WebCenter Sites instances yet.

10:18 am: Synchronize the Workspace to the WebCenter Sites Instance

Sonoko opens the Developer Tools Synchronization and selects the **Import** tab. All resources contained in Sonoko's main Developer Tools workspace are listed.

As required, she will first import the site definition, then the flex family, and then the assets, in separate runs as described below:

> **Note:**
>
> Matthäus will do the same later, when he finishes his meeting with Marketing.

1. Import the site definition (Acceptance in this scenario).

Sonoko imports the site definition first. She narrows down her search by using the `Site.*Accepta` expression in the search field. She then selects the site (Acceptance) and synchronizes it to her WebCenter Sites instance by clicking **Sync to WebCenter Sites** .

Figure 36-11   Importing the Site Definition
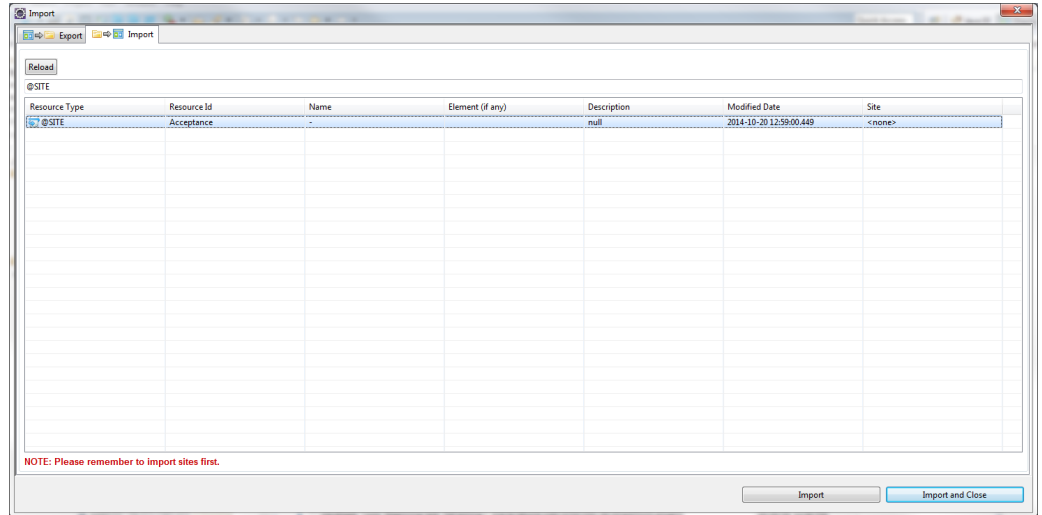
Using the Sites Log view, Sonoko verifies that the site is imported successfully.

Figure 36-12   Confirmation of Site Import
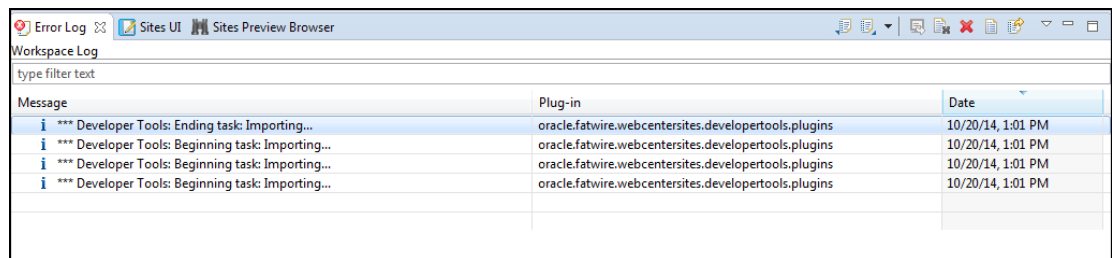
2. Sonoko opens the synchronization configuration again, and imports the site's flex family, starting with the flex attribute (`Author_A` in this scenario).

Because Sonoko did not set up the Acceptance site's flex family on her WebCenter Sites instance, she must first import the flex attribute (`Author_A`) to her WebCenter Sites instance. After the flex attribute is imported, she can then synchronize the rest of the asset types that comprise the site's flex family to her WebCenter Sites instance.

Figure 36-13   Importing Flex Attributes
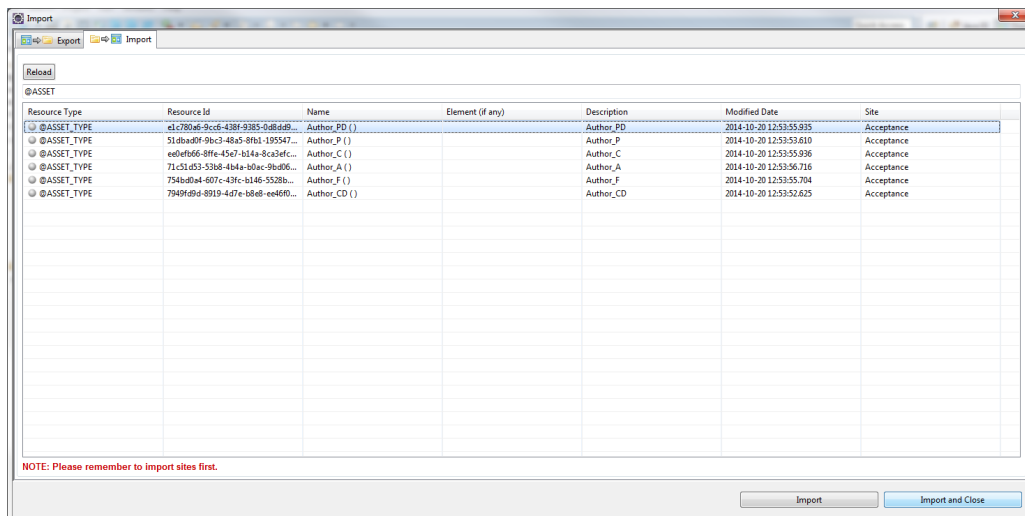
3. As a final step, Sonoko synchronizes the flex definition, which automatically imports the required attributes.

   The Sites Log view shows that the local asset identifiers of all the site's resources are re-mapped when imported into the new WebCenter Sites instance.

Figure 36-14   Synchronizing the Flex Definition
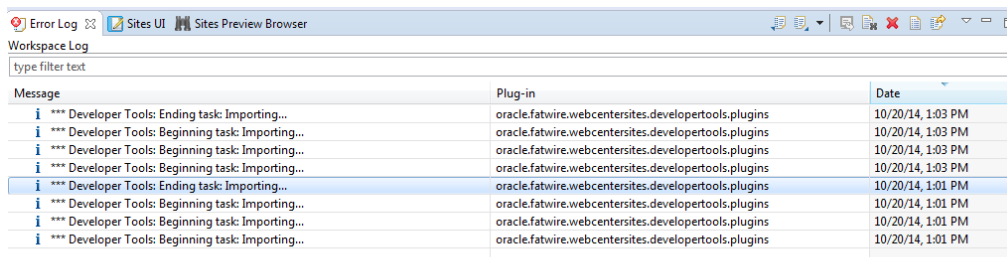
10:21 am: Assign Site Permissions

After synchronizing the resources to her WebCenter Sites instance, Sonoko assigns site permissions to herself. These permissions enable her to access the site and its resources from the Admin interface.

Note:

To access the tree applet in the new site, Sonoko must assign at least one tree tab to the site.

Figure 36-15   Assigning Site Permissions
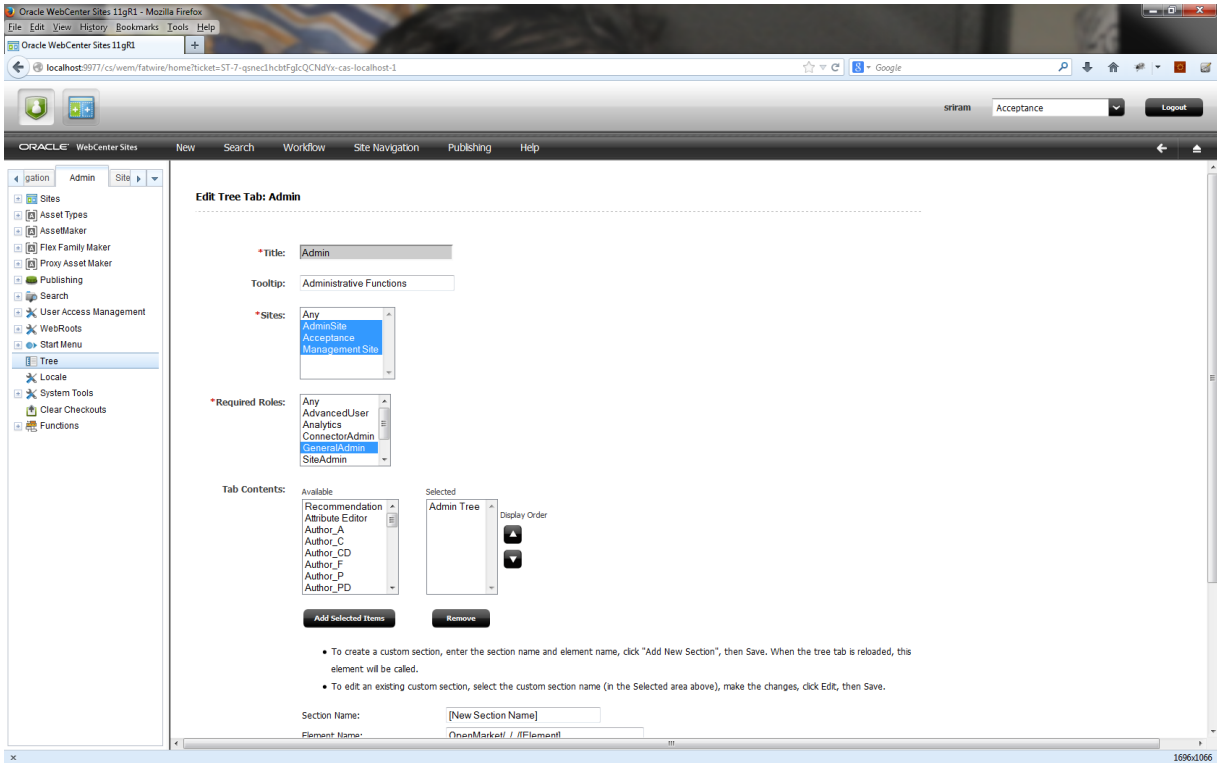
10:22 am: The Start Menu Issue

Sonoko logs into the site, and clicks the **New** option. However, she finds there are no start menu items available. Of course, Artie did not check the site's start menu items into the SVN repository.

Figure 36-16   No Start Menu Items Available
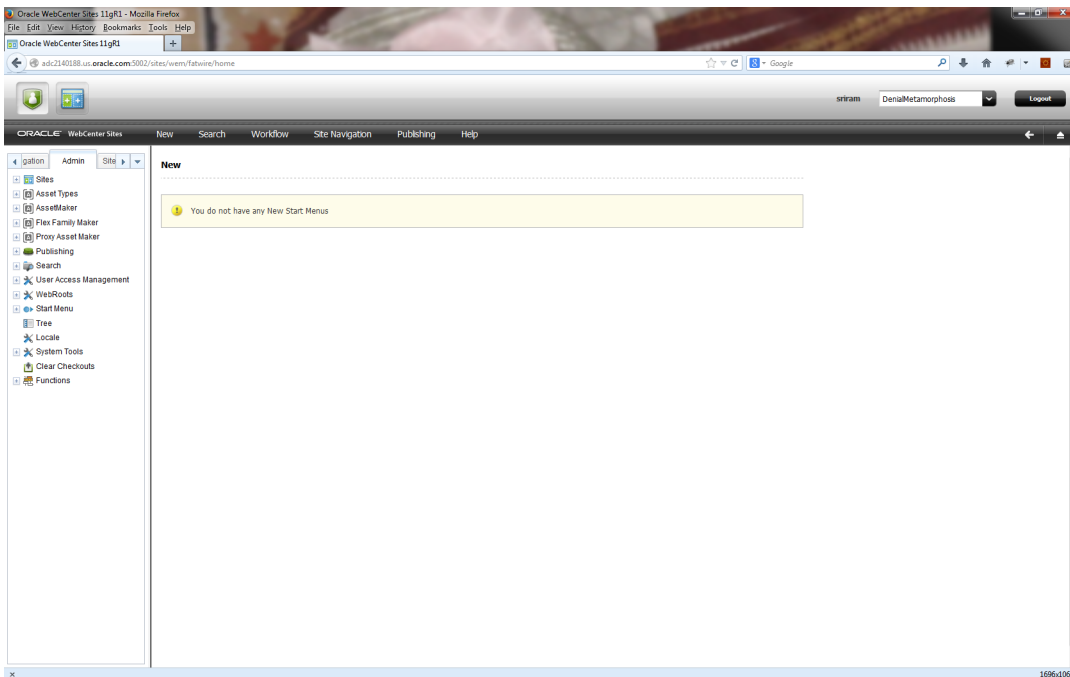
10:24 am: Resolving the Start Menu Issue

Sonoko sends Artie an IM informing him that he forgot to check-in the new site's start menu items.

1. Artie synchronizes the site's start menu items to his main Developer Tools workspace.

Figure 36-17 Synchronizing Start Menu Items to Developer Tools Workspace
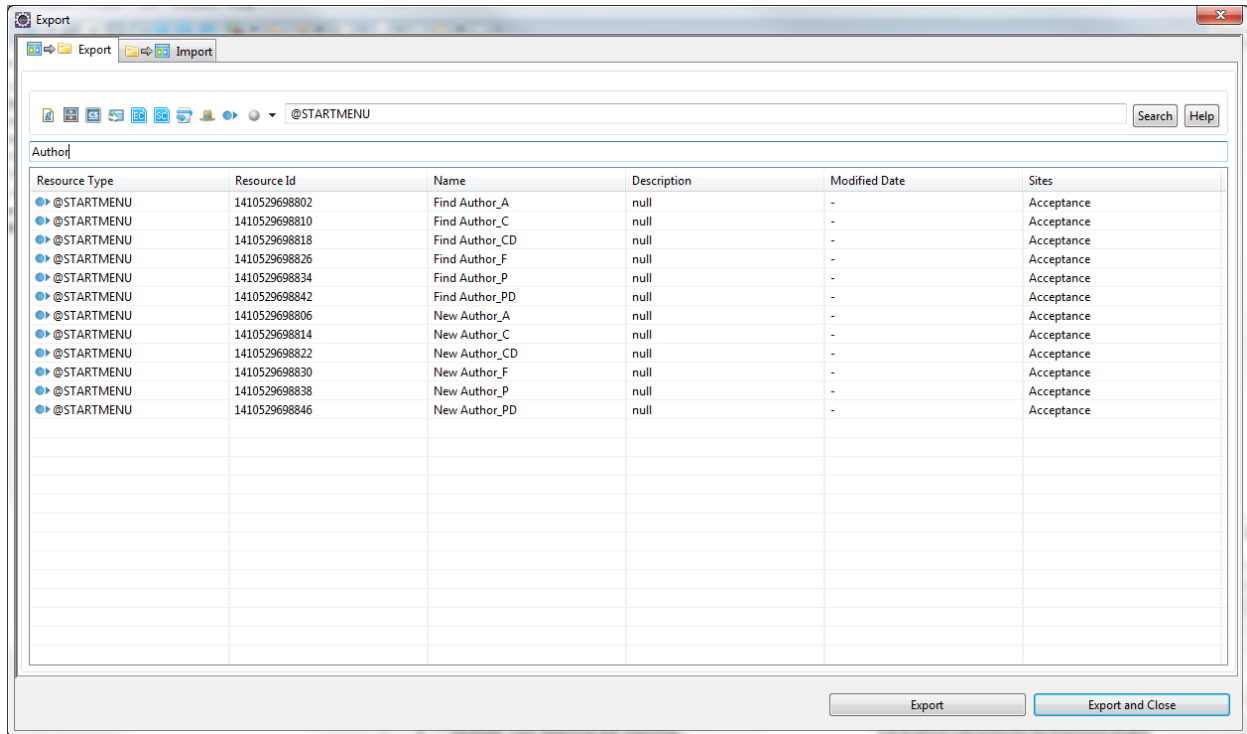
2. Artie then checks the site's start menu items into the SVN repository.

Figure 36-18   Committing Start Menu Items to SVN
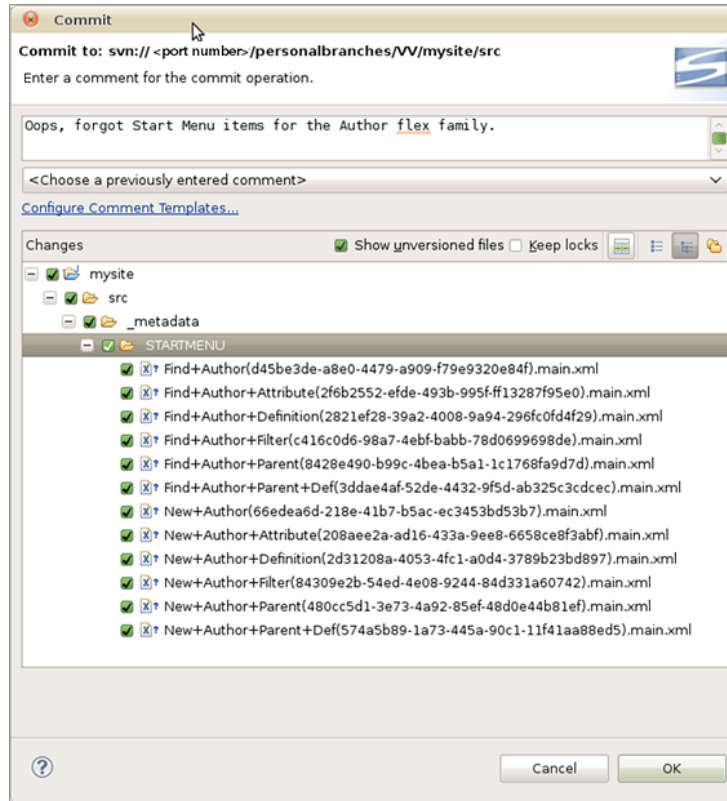
3. Sonoko finds that Artie committed the start menu items to the SVN. Sonoko then updates her Eclipse project. She accesses the SVN repository and synchronizes the start menu items to her main Developer Tools workspace. She then imports those start menu items to her WebCenter Sites instance.

Figure 36-19   Importing Start Menu Items to WebCenter Sites Instance
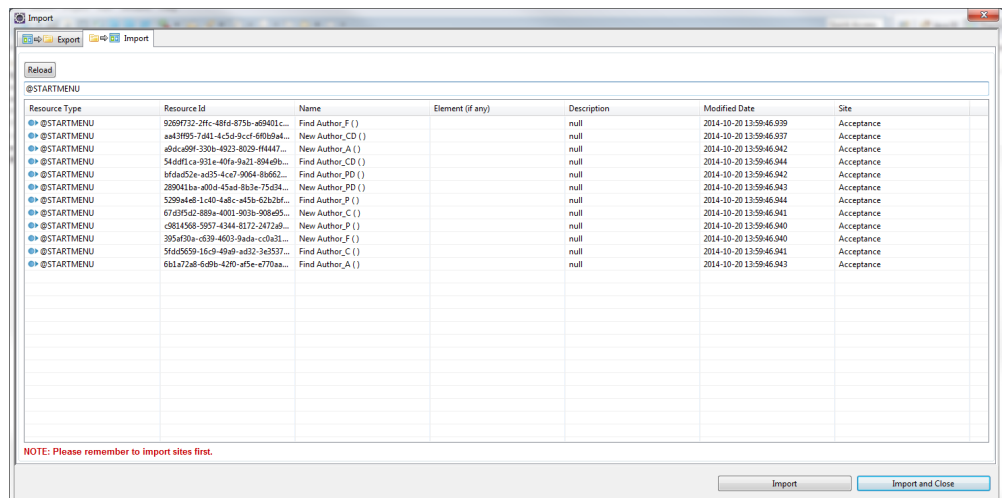
4. Without restarting her WebCenter Sites instance, Sonoko clicks **Search**.

    The start menu items she imported into her WebCenter Sites instance are listed.

Figure 36-20 Start Menu Items List
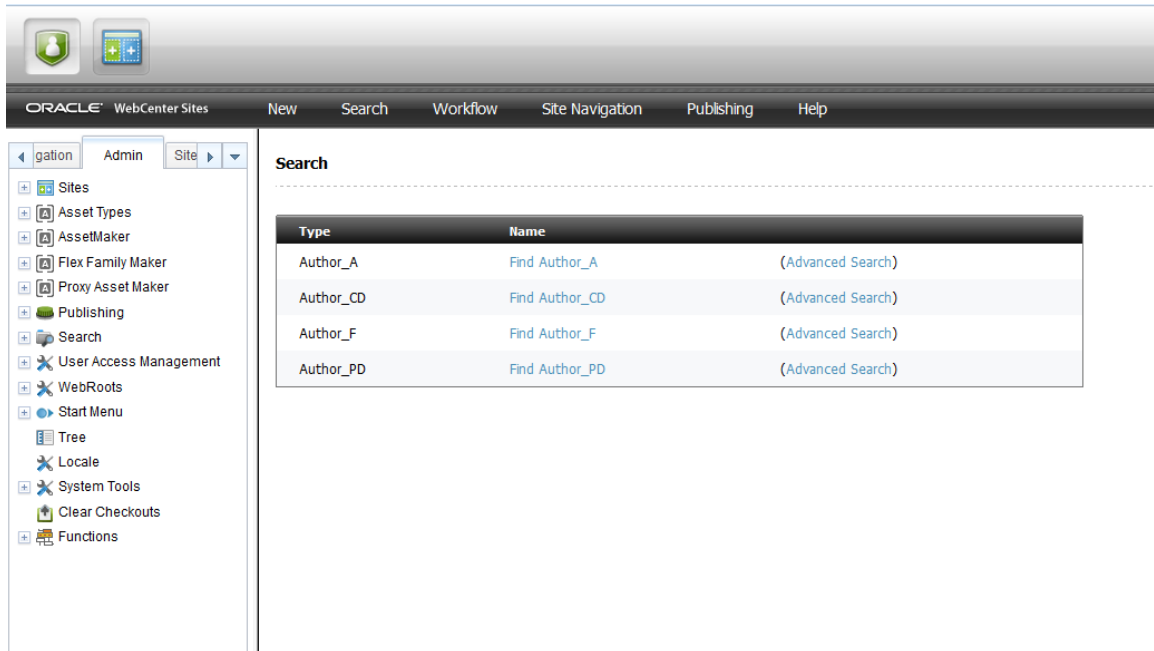
11:17 am: Marketing Requests Changes

```
Subject: Proposed Author Definition Changes

Date: Wed, 16 Feb 2011 11:17:39

From: matthäus.companynone.com

To: Tech-Development

Team,

I just synchronized your changes into my system. As per my
meeting with Marketing, we must have date of birth and
birthplace attributes in the Author Definition. I noticed these
attributes do not exist, so I will add them. Artie, can you
review the changes I make when you have the chance?

Regards,

Matthäus
```

11:22 am: Adding New Attributes to the Author Definition

Matthäus creates the attributes Marketing requested and adds them to the flex definition (Author definition in this scenario). He then exports the new attributes and the flex definition to his main Developer Tools workspace and commits them to the SVN repository.

Figure 36-21   Exporting New Attributes and Flex Definition
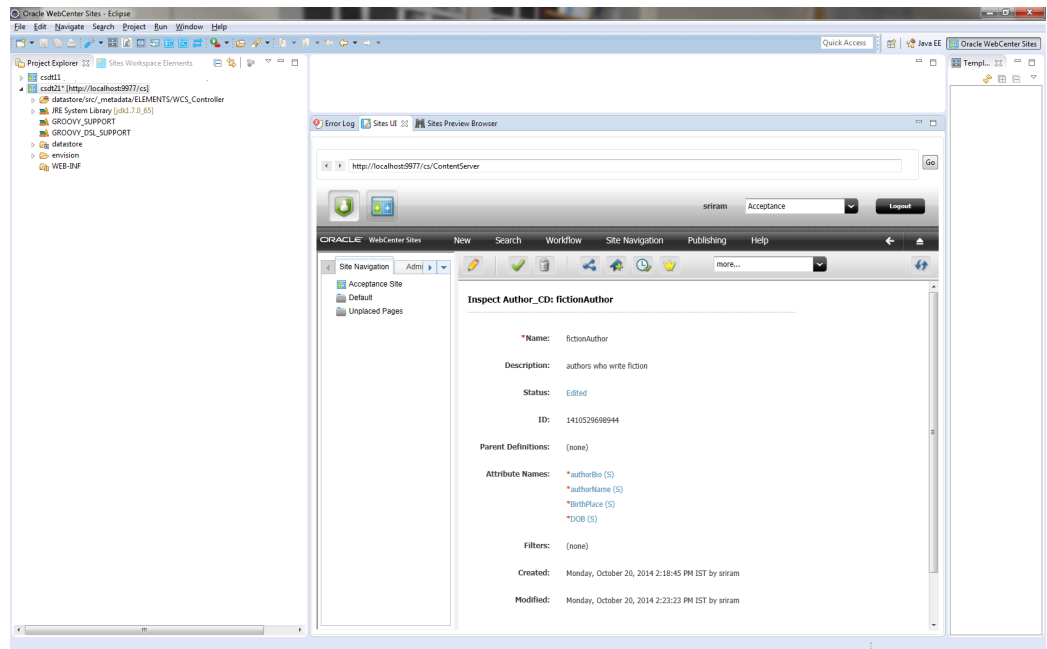
11:25 am: Reviewing the Changes to the Site

Artie retrieves the modified Author definition from SVN and imports it into his WebCenter Sites instance.

Figure 36-22   Reviewing Changes to the Site
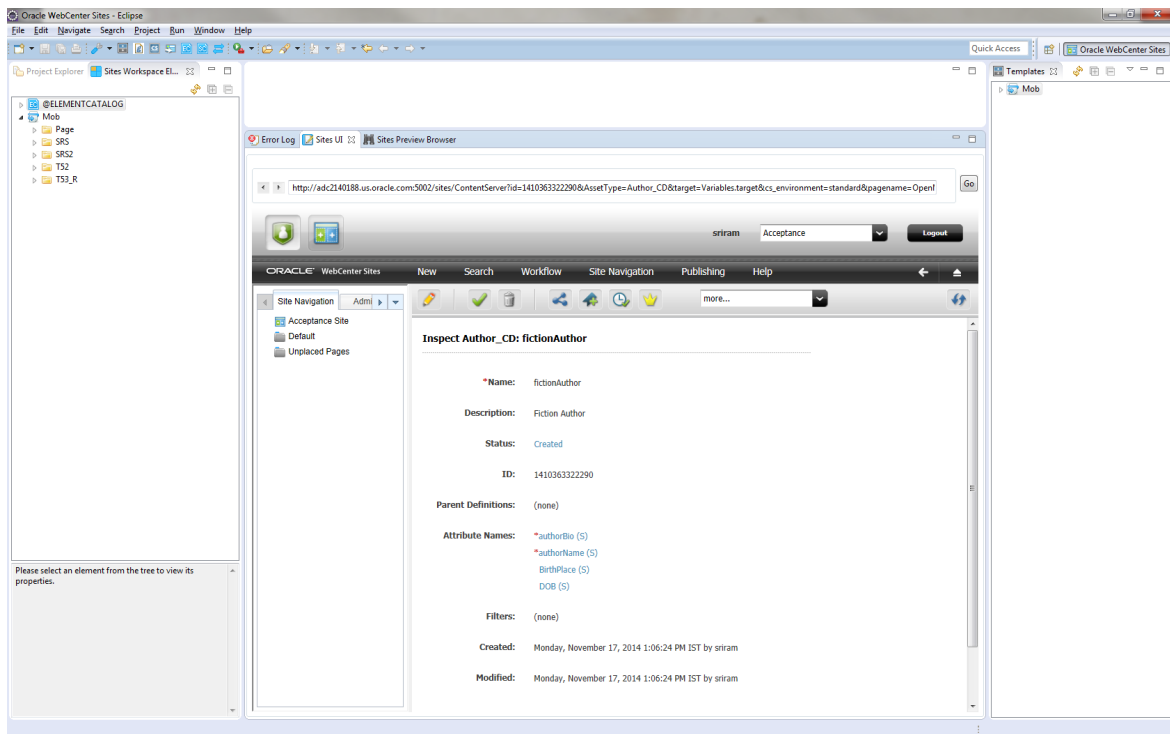

```
Subject: RE: Proposed Author Definition Changes

Date: Wed, 16 Feb 2011 11:37:31
```

```
From: artie.companynone.com

To: matthäus.companynone.com

Matthäus,

Thank you for taking care of this. Corporate standards require
us to capitalize the first letter of each subsequent word. I
will delete the birthplace attribute and add birthPlace instead.

Thank you,

Artie
```

11:44 am: Modifying the Attributes of the Author Definition

1. Artie creates the `birthPlace` attribute and adds it to the flex definition. He then removes the original `birthplace` attribute from the site definition.

2. Artie commits the new attribute and the changes to the Author definition to the SVN repository. He then verifies that the `birthplace` attribute has a status of `VO`, indicating the attribute is voided.

   When Sonoko and Matthäus update their WebCenter Sites instances, the `birthplace` attribute is correspondingly voided on their own workspaces.

Figure 36-23   Modifying Attributes of the Author Definition
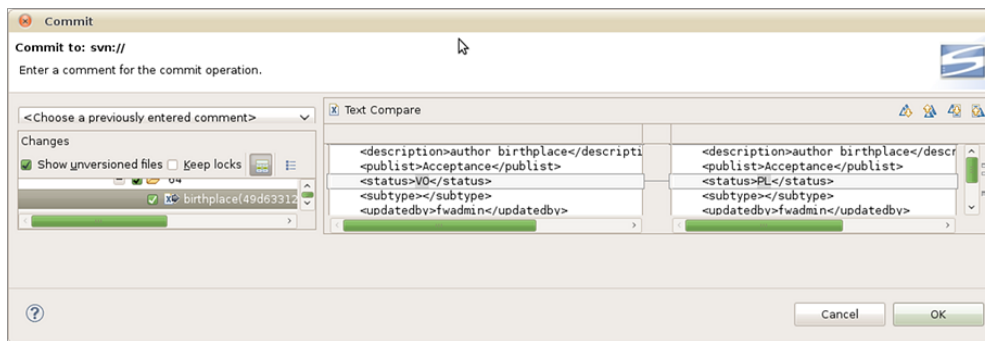
11:53 am: The Team Updates Their Workspaces and WebCenter Sites Instances

1. Sonoko and Matthäus update their main Developer Tools workspaces with the resources Artie checked in to the SVN repository.

2. They then import the resources in their workspaces to their WebCenter Sites instances by opening the **Import** tab. For convenience, they sorted by the `Modified Date` column so the most recent changes are shown on top.

   Any voided attributes (such as the `birthplace` attribute Artie voided) show a status hint (`status=VO`) in the `Name` column.

Figure 36-24   Updating Workspaces and WebCenter Sites Instances
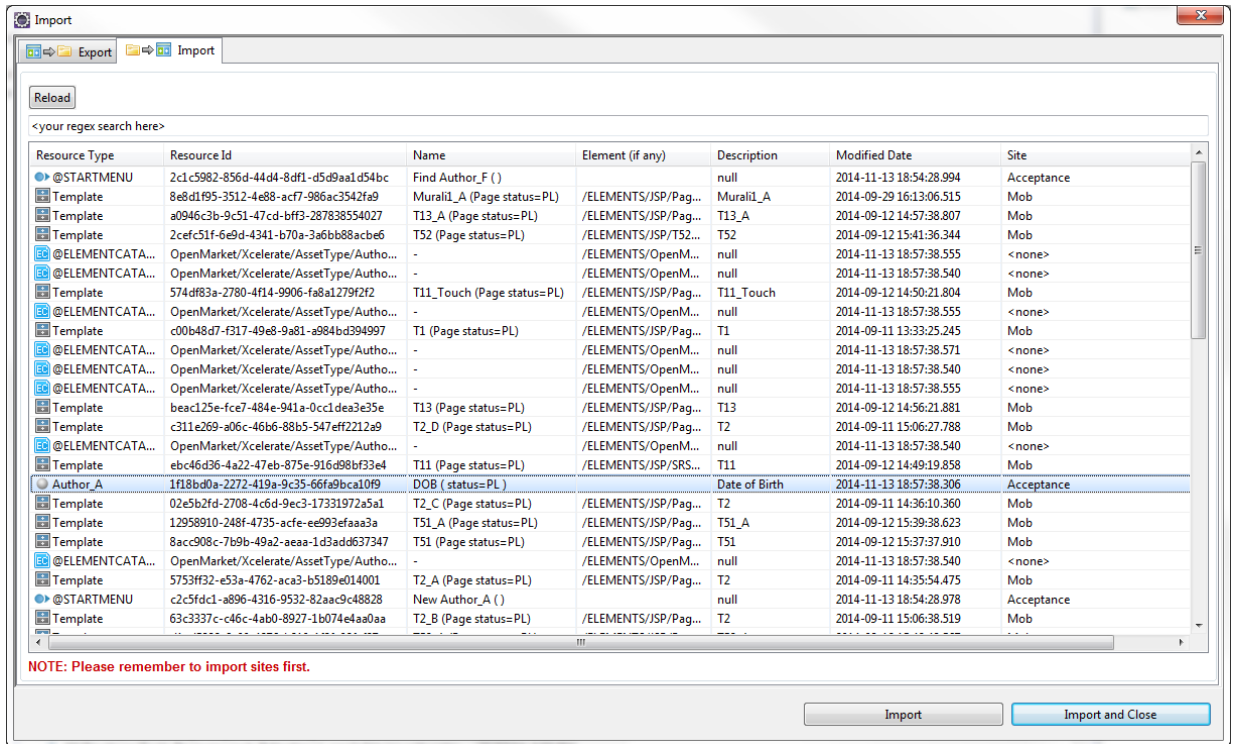
3.   Sonoko and Matthäus import these changes from their workspaces to their
      WebCenter Sites instances. Their workspaces and WebCenter Sites instances are
      now up to date.

12:27 pm: The Team Creates a Template Asset for the Site

1.   (12:27 pm) Matthäus creates a Template asset for the site's **Welcome** page.

Figure 36-25   Creating Template Asset for Site
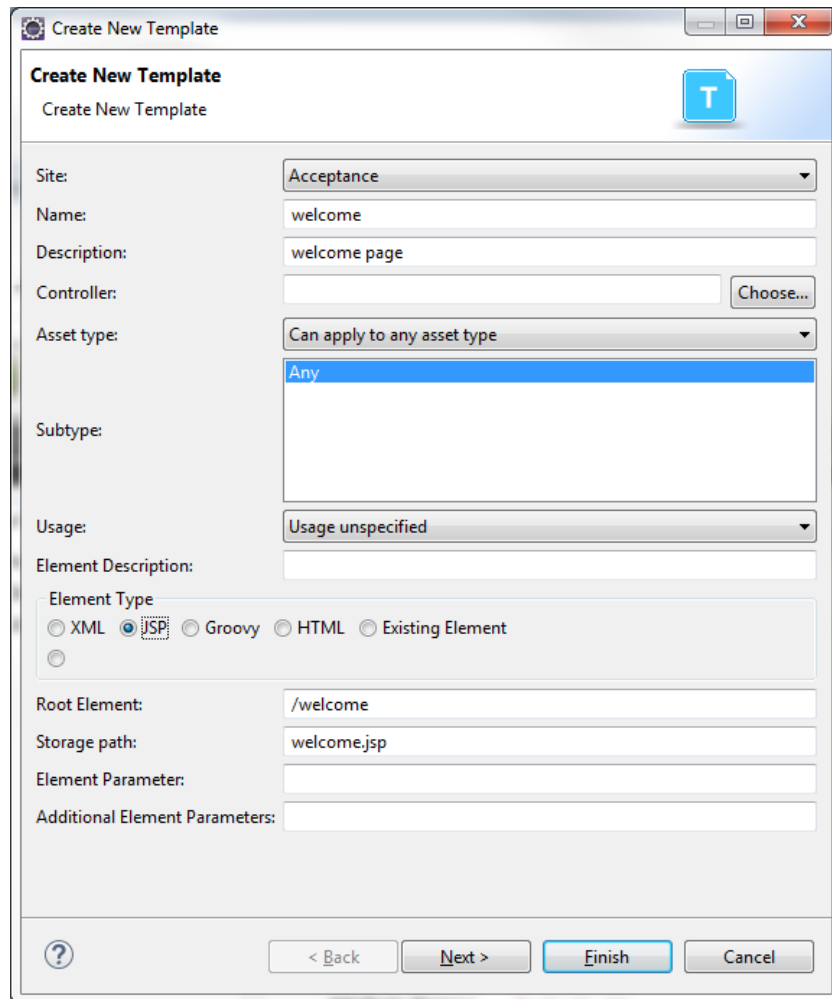
2. (12:34 pm) Matthäus edits the Template asset and previews the changes in the Sites Preview Browser view. As soon as he saves the changes made to the Template asset's JSP, he uses the **Ctrl+R** keyboard command to refresh the preview browser.

Figure 36-26   Sites Preview Browser Edits
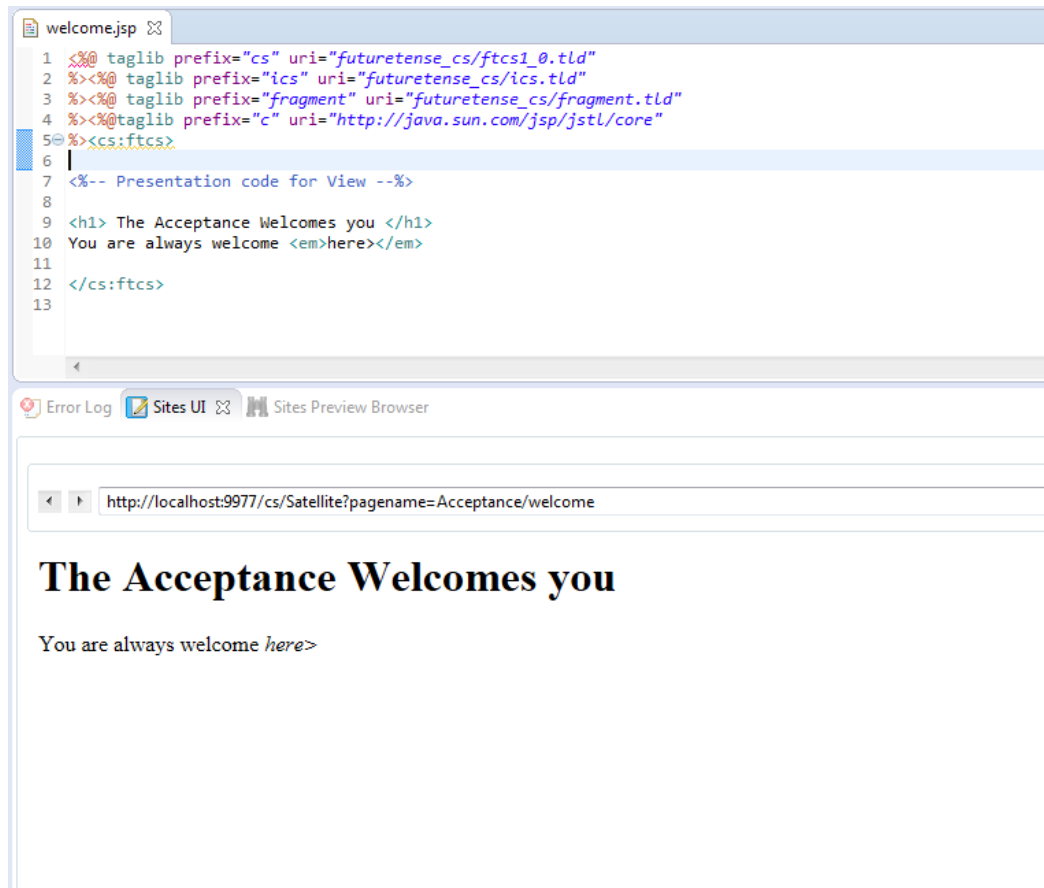
3. (12:39 pm) Matthäus commits the Template's `.jsp` and `.main.xml` files to the SVN repository.

    Subclipse finds all changes to the project and brings those changes to the attention of the developer. Because the only new asset was the Template asset, Matthäus is able to deduce that the `.main.xml` file is the Template's metadata and the JSP file is the Template's code.

Figure 36-27   Committed Template Files
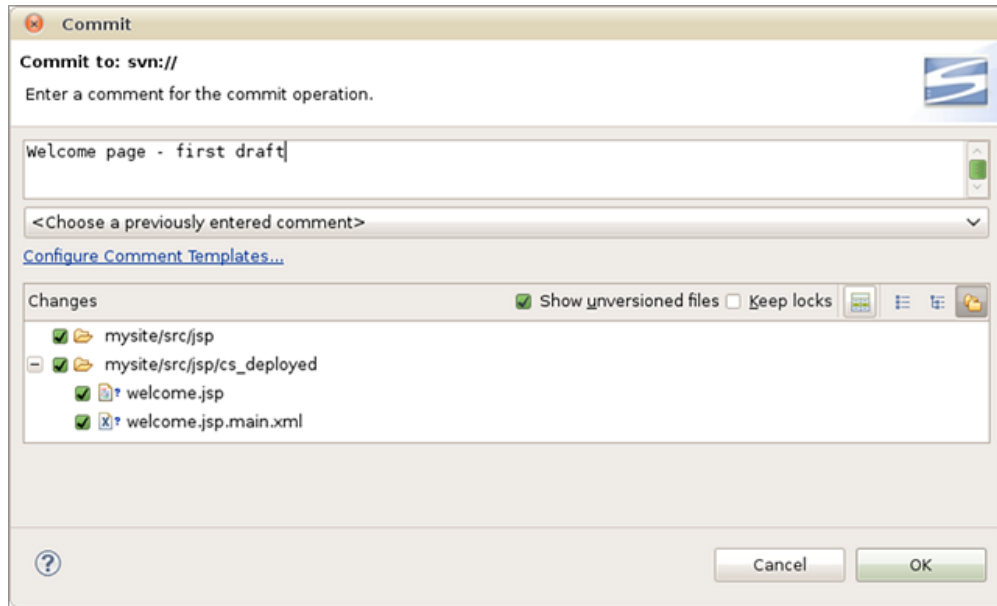
4. (12:44 pm) Sonoko makes some touch ups to the Template's JSP file in her own workspace.

Figure 36-28   Editing the Templates JSP File in Workspace
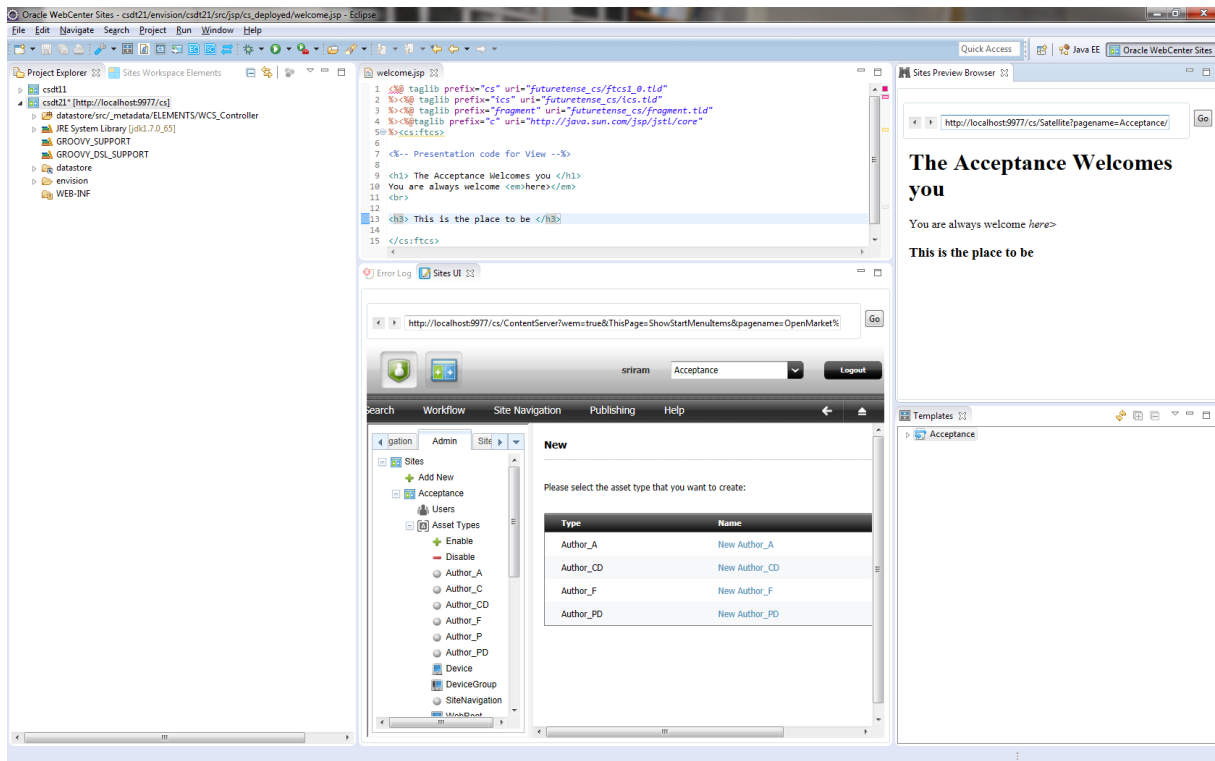
5. Sonoko reviews the changes to the JSP file and then commits those changes to the SVN.

Figure 36-29   Committing Changes to SVN
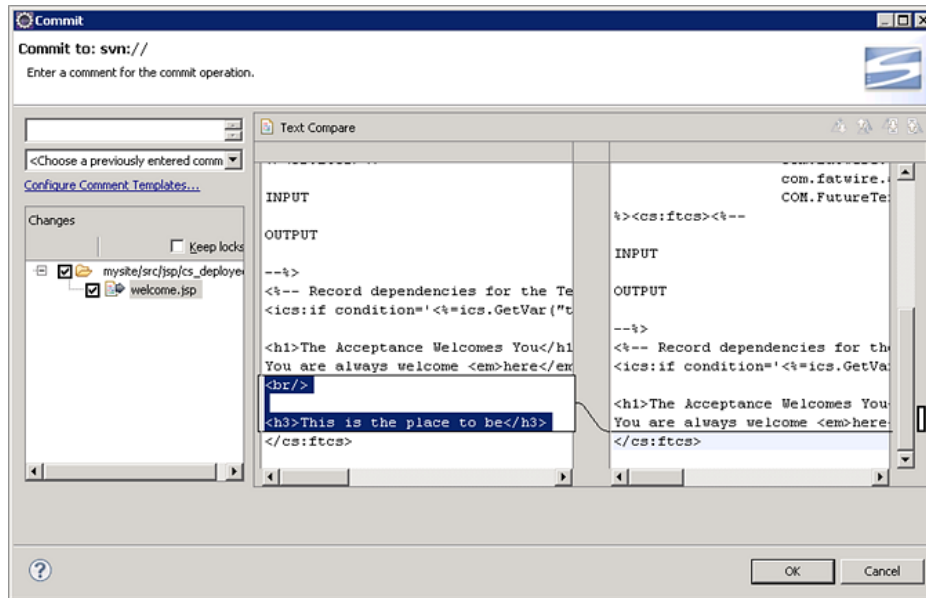
> **Note:**
>
> If another team member were to modify and check-in this file at the same time as Sonoko, SVN would indicate to Sonoko that another version of the file is checked in. She would then be able to integrate those changes with her own to avoid inadvertent overwrites.

36.2 Three Days Later... Deployment

Yogesh uses the command line interface to deploy the site.

See Using Developer Tools Command-Line Tool.

9:32 am: Preparing for Deployment

Yogesh finally got around to setting up the test environment and is preparing to deploy the current build using the command line interface. He installed a WebCenter Sites system on hardware that matches the environment used in production.

To test the Developer Tools import before adding it to a fully-automated nightly script:

1. Using the command line interface, Yogesh checks the Acceptance site and its resources out of SVN and into the workspace of the target WebCenter Sites instance.

    Command:

    ```
    ## go to the workspace location under export/envision/
       cs_workspace in the ${sites-shared}/config directory

    ## checkout site from svn
    ${sites-shared}/config/export/envision/cs_workspace$ svn checkout  svn://
       yoursvnhost/projects/mysite/src
    ```

    Output:

```
A    mysite/src
A    mysite/src/_metadata
A    mysite/src/_metadata/ASSET
A    mysite/src/_metadata/ASSET/Author_A
A    mysite/src/_metadata/ASSET/Author_A/10
A    mysite/src/_metadata/ASSET/Author_A/10/14
A    mysite/src/_metadata/ASSET/Author_A/10/14/authorName(cbf4d8aa-d23a-4f0d-
b55d-a87a0e9bbf33).main.xml
A    mysite/src/_metadata/ASSET/Author_A/11
A    mysite/src/_metadata/ASSET/Author_A/11/79
A    mysite/src/_metadata/ASSET/Author_A/11/79/birthPlace(42afd458-e90c-4e18-
a4b6-47d322b46414).main.xml
A    mysite/src/_metadata/ASSET/Author_A/5
A    mysite/src/_metadata/ASSET/Author_A/5/64
A    mysite/src/_metadata/ASSET/Author_A/5/64/birthplace(49d63312-c74d-4ccd-
bb7f-4dc698a9da22).main.xml
A    mysite/src/_metadata/ASSET/Author_A/15
A    mysite/src/_metadata/ASSET/Author_A/15/76
A    mysite/src/_metadata/ASSET/Author_A/15/76/
DOB(9fe04c6e-36e7-4ee3-8c76-8c02edf74136).main.xml
A    mysite/src/_metadata/ASSET/Author_A/71
A    mysite/src/_metadata/ASSET/Author_A/71/74
A    mysite/src/_metadata/ASSET/Author_A/71/74/authorBio(ada2d6be-ef14-4766-
b446-911bfa838835).main.xml
A    mysite/src/_metadata/ASSET/Author_CD
A    mysite/src/_metadata/ASSET/Author_CD/76
A    mysite/src/_metadata/ASSET/Author_CD/76/4
A    mysite/src/_metadata/ASSET/Author_CD/76/4/fictionAuthor(71d6067b-35f6-47f4-
ae97-3876303abb37).main.xml
A    mysite/src/_metadata/ASSET_TYPE
A    mysite/src/_metadata/ASSET_TYPE/Author_F(5f9b4964-e9be-4f25-
a413-877e8a5c7469).main.xml
A    mysite/src/_metadata/ASSET_TYPE/
Author_P(1552d907-5f38-400b-9460-36e46d14abc3).main.xml
A    mysite/src/_metadata/ASSET_TYPE/Author_A(162d0b70-7e69-4266-
acca-2f472e3d71bd).main.xml
A    mysite/src/_metadata/ASSET_TYPE/
Author_CD(33faf87e-9e8f-4f49-97cd-424810408938).main.xml
A    mysite/src/_metadata/ASSET_TYPE/Author_PD(7c748df3-
d149-4b71-802a-64b11360e74b).main.xml
A    mysite/src/_metadata/ASSET_TYPE/Author_C(d1497b50-665c-4b0c-80a7-
d25f61566be4).main.xml
A    mysite/src/_metadata/STARTMENU
A    mysite/src/_metadata/STARTMENU/Find+Author+Attribute(2f6b2552-
efde-493b-995f-ff13287f95e0).main.xml
...
```

2. Yogesh runs a workspace listing (cmd=listds) to verify that the site and all of its resources will be imported into the WebCenter Sites instance. He uses the @ALL_ASSETS and @ALL_NONASSETS selectors to generate listings of all asset and non-asset resources in the workspace:

- Command to use the @ALL_ASSETS selector:

```
${sites-shared}/config/export/envision/cs_workspace$
java - Xbootclasspath/a:lib/servlet-api.jar -jar
developer-tools-command-line.jar http://<host>:<port>/
<context>/ContentServer username=fwadmin
password=xceladmin resources=@ALL_ASSETS cmd=listds
```

Output:

```
Resource Type |||   Resource Id   |||   Name   |||
Description   ||| Modified On
------------------------------------------------------------------
Author_A ||| cbf4d8aa-d23a-4f0d-b55d-a87a0e9bbf33 ||| authorName
( status=ED ) ||| author name ||| 2011-02-17 15:26:34.000
Author_A ||| 42afd458-e90c-4e18-a4b6-47d322b46414 ||| birthPlace
( status=PL ) ||| place of birth ||| 2011-02-17 15:26:34.000
Author_A ||| 9fe04c6e-36e7-4ee3-8c76-8c02edf74136 ||| DOB
( status=PL ) ||| date of birth ||| 2011-02-17 15:26:34.000
Author_CD ||| 71d6067b-35f6-47f4-ae97-3876303abb37 |||
fictionAuthor ( status=ED ) ||| authors who write fiction |||
2011-02-17 15:26:34.000
Author_A ||| ada2d6be-ef14-4766-b446-911bfa838835 ||| authorBio
( status=ED ) ||| author biography ||| 2011-02-17 15:26:34.000
Author_A ||| 49d63312-c74d-4ccd-bb7f-4dc698a9da22 ||| birthplace
( status=VO ) ||| author birthplace ||| 2011-02-17 15:12:43.000
Template ||| 89b05c0f-227b-4dcb-961e-2ab6e6af2dae ||| welcome
(Typeless status=PL) ||| welcome page ||| 2011-02-17 23:18:18.000
```

- Command to use the `@ALL_NONASSETS` selector:

```
${sites-shared}/config/export/envision/cs_workspace$
        java - Xbootclasspath/a:lib/servlet-api.jar -jar
        developer-tools-command-line.jar http://<host>:<port>/
        <context>/ContentServer username=fwadmin
        password=xceladmin resources=@ALL_NONASSETS cmd=listds
```

Output:

```
Resource Type |||   Resource Id   |||   Name   |||
Description   ||| Modified On
------------------------------------------------------------------
@STARTMENU ||| 66edea6d-218e-41b7-b5ac-ec3453bd53b7 ||| New
Author ( ) ||| null ||| 2011-02-18 11:02:23.000
@STARTMENU ||| c416c0d6-98a7-4ebf-babb-78d0699698de ||| Find
Author Filter ( ) ||| null ||| 2011-02-18 11:02:23.000
@ASSET_TYPE ||| 162d0b70-7e69-4266-acca-2f472e3d71bd ||| Author_A
( ) ||| Author Attribute ||| 2011-02-18 11:02:23.000
@STARTMENU ||| 2821ef28-39a2-4008-9a94-296fc0fd4f29 ||| Find
Author Definition ( ) ||| null ||| 2011-02-18 11:02:23.000
@STARTMENU ||| d45be3de-a8e0-4479-a909-f79e9320e84f ||| Find
Author ( ) ||| null ||| 2011-02-18 11:02:23.000
@STARTMENU ||| 2f6b2552-efde-493b-995f-ff13287f95e0 ||| Find
Author Attribute ( ) ||| null ||| 2011-02-18 11:02:23.000
@ASSET_TYPE ||| 7c748df3-d149-4b71-802a-64b11360e74b ||| Author_
PD ( ) ||| Author Parent Def ||| 2011-02-18 11:02:23.000
@STARTMENU ||| 208aee2a-ad16-433a-9ee8-6658ce8f3abf ||| New
Author Attribute ( ) ||| null ||| 2011-02-18 11:02:23.000
@STARTMENU ||| 8428e490-b99c-4bea-b5a1-1c1768fa9d7d ||| Find
Author Parent ( ) ||| null ||| 2011-02-18 11:02:23.000
@ASSET_TYPE ||| d1497b50-665c-4b0c-80a7-d25f61566be4 ||| Author_C
( ) ||| Author ||| 2011-02-18 11:02:23.000
…
```

3. Yogesh then makes sure all necessary asset types will be imported by using the
   `@ASSET_TYPE:*` selector:

Command:

```
${sites-shared}/config/export/envision/cs_workspace$
    java - Xbootclasspath/a:lib/servlet-api.jar -jar
    developer-tools-command-line.jar http://<host>:<port>/
```

```
                                <context>/ContentServer username=fwadmin password=xceladmin
                                resources=@ASSET_TYPE:* cmd=listds
```

Output:

```
Resource Type |||    Resource Id    |||    Name     |||   Description   ||| Modified On
--------------------------------------------------------------------------------------
Author_A ||| cbf4d8aa-d23a-4f0d-b55d-a87a0e9bbf33 ||| authorName ( status=ED ) ||| author name
||| 2011-02-17 15:26:34.000
Author_A ||| 42afd458-e90c-4e18-a4b6-47d322b46414 ||| birthPlace ( status=PL ) ||| place of birth
||| 2011-02-17 15:26:34.000
Author_A ||| 9fe04c6e-36e7-4ee3-8c76-8c02edf74136 ||| DOB ( status=PL ) ||| date of birth |||
2011-02-17 15:26:34.000
Author_CD ||| 71d6067b-35f6-47f4-ae97-3876303abb37 ||| fictionAuthor ( status=ED ) ||| authors
who write fiction ||| 2011-02-17 15:26:34.000
Author_A ||| ada2d6be-ef14-4766-b446-911bfa838835 ||| authorBio ( status=ED ) ||| author
biography ||| 2011-02-17 15:26:34.000
Author_A ||| 49d63312-c74d-4ccd-bb7f-4dc698a9da22 ||| birthplace ( status=VO ) ||| author
birthplace ||| 2011-02-17 15:12:43.000
Template ||| 89b05c0f-227b-4dcb-961e-2ab6e6af2dae ||| welcome (Typeless status=PL) ||| welcome
page ||| 2011-02-17 23:18:18.000
```

4. Yogesh notes that all necessary resources for the site will be imported into the build.

10:04 am: Deploying the Site and its Resources

Using the command line interface, Yogesh runs the import sequence.

1. First, he imports the site:

   Command:

   ```
   ${sites-shared}/config/export/envision/cs_workspace$
    java - Xbootclasspath/a:lib/servlet-api.jar -jar
      developer-tools-command-line.jar http://<host>:<port>/
        <context>/ContentServer username=fwadmin
          password=xceladmin resources=@SITE:Acceptance cmd=import
   ```

   Output:

   ```
    *** Importing batch 1297868431526
   Importing DSKEY @SITE-Acceptance (batch 1297868431526)
   Saved Acceptance (batch 1297868431526)
   *** Completed importing batch 1297868431526
   ```

2. Then, the flex family:

   Command:

   ```
   ${sites-shared}/config/export/envision/cs_workspace$
   java - Xbootclasspath/a:lib/servlet-api.jar -jar
     developer-tools-command-line.jar http://<host>:<port>/
       <context>/ContentServer username=fwadmin
         password=xceladmin resources=@ASSET_TYPE:* cmd=import
   ```

   Output:

   ```
   *** Importing batch 1298064678765
   Importing DSKEY @ASSET_TYPE-162d0b70-7e69-4266-acca-2f472e3d71bd (batch
   1298064678765)
   Importing DSKEY @ELEMENTCATALOG-OpenMarket/Xcelerate/AssetType/Author_A/LoadTree
   (batch 1298064678765)
   ```

```
Saved OpenMarket/Xcelerate/AssetType/Author_A/LoadTree (batch 1298064678765)
…
```

3. Next, the assets:

Command:

```
${sites-shared}/config/export/envision/cs_workspace
java - Xbootclasspath/a:lib/servlet-api.jar -jar
  developer-tools-command-line.jar http://<host>:<port>/
   <context>/ContentServer username=fwadmin
     password=xceladmin resources=@ALL_ASSETS cmd=import
```

Output:

```
*** Importing batch 1298064679760
Importing DSKEY Author_A-cbf4d8aa-d23a-4f0d-b55d-a87a0e9bbf33 (batch 1298064679760)
Dependency @ASSET_TYPE-Author_A already exists, skipping.
Saved Author_A:1295889071437 (batch 1298064679760)
Importing DSKEY Author_A-42afd458-e90c-4e18-a4b6-47d322b46414 (batch 1298064679760)
Dependency @ASSET_TYPE-Author_A already exists, skipping.
Saved Author_A:1295889071441 (batch 1298064679760)
Importing DSKEY Author_A-9fe04c6e-36e7-4ee3-8c76-8c02edf74136 (batch 1298064679760)
Dependency @ASSET_TYPE-Author_A already exists, skipping.
Saved Author_A:1295889071445 (batch 1298064679760)
Importing DSKEY Author_CD-71d6067b-35f6-47f4-ae97-3876303abb37 (batch 1298064679760)
Importing DSKEY Author_A-ada2d6be-ef14-4766-b446-911bfa838835 (batch 1298064679760)
Dependency @ASSET_TYPE-Author_A already exists, skipping.
Saved Author_A:1295889071449 (batch 1298064679760)
Dependency @ASSET_TYPE-Author_C already exists, skipping.
Dependency @ASSET_TYPE-Author_P already exists, skipping.
Dependency @ASSET_TYPE-Author_CD already exists, skipping.
Dependency @ASSET_TYPE-Author_PD already exists, skipping.
Dependency @ASSET_TYPE-Author_F already exists, skipping.
Dependency @ASSET_TYPE-Author_A already exists, skipping.
Saved Author_CD:1295889071453 (batch 1298064679760)
Importing DSKEY Author_A-49d63312-c74d-4ccd-bb7f-4dc698a9da22 (batch 1298064679760)
Dependency @ASSET_TYPE-Author_A already exists, skipping.
Saved Author_A:1295889071460 (batch 1298064679760)
Importing DSKEY Template-89b05c0f-227b-4dcb-961e-2ab6e6af2dae (batch 1298064679760)
Saved Template:1295889071461 (batch 1298064679760)
*** Completed importing batch 1298064679760
```

4. Because this is a delivery install, start menu items are optional. However, Yogesh imports the start menu items because he wants to use the Admin interface to verify that all of the resources are successfully imported.

Command:

```
${sites-shared}/config/export/envision/cs_workspace$
java - Xbootclasspath/a:lib/servlet-api.jar -jar
  developer-tools-command-line.jar http://<host>:<port>/
   <context>/ContentServer username=fwadmin password=xceladmin
     resources=@STARTMENU:* cmd=import
```

Output:

```
*** Importing batch 1298064681075
Importing DSKEY @STARTMENU-66edea6d-218e-41b7-b5ac-ec3453bd53b7 (batch 1298064681075)
Saved 1297720502210 (batch 1298064681075)
Importing DSKEY @STARTMENU-c416c0d6-98a7-4ebf-babb-78d0699698de (batch 1298064681075)
Saved 1297720502230 (batch 1298064681075)
Importing DSKEY @STARTMENU-2821ef28-39a2-4008-9a94-296fc0fd4f29 (batch 1298064681075)
```

```
Saved 1297720502222 (batch 1298064681075)
Importing DSKEY @STARTMENU-d45be3de-a8e0-4479-a909-f79e9320e84f (batch 1298064681075)
Saved 1297720502206 (batch 1298064681075)
Importing DSKEY @STARTMENU-2f6b2552-efde-493b-995f-ff13287f95e0 (batch 1298064681075)
Saved 1297720502214 (batch 1298064681075)
Importing DSKEY @STARTMENU-208aee2a-ad16-433a-9ee8-6658ce8f3abf (batch 1298064681075)
Saved 1297720502218 (batch 1298064681075)
Importing DSKEY @STARTMENU-8428e490-b99c-4bea-b5a1-1c1768fa9d7d (batch 1298064681075)
Saved 1297720502238 (batch 1298064681075)
Importing DSKEY @STARTMENU-2d31208a-4053-4fc1-a0d4-3789b23bd897 (batch 1298064681075)
Saved 1297720502226 (batch 1298064681075)
Importing DSKEY @STARTMENU-480cc5d1-3e73-4a92-85ef-48d0e44b81ef (batch 1298064681075)
Saved 1297720502242 (batch 1298064681075)
Importing DSKEY @STARTMENU-84309e2b-54ed-4e08-9244-84d331a60742 (batch 1298064681075)
*** Completed importing batch 1298064681075
```

10:55 am: The Deployment is Successful

Yogesh concludes that the import sequence was successful. He plans to automate daily installs on this system by writing the following script:

```
## Reinstall ContentServer to start with a clean slate.
## Optionally skip this and just do an update
Reinstall_CS()

## Bring in the latest source from SVN
SVN_Update()

## Prepare for import: compile any Java code such as url assemblers and flex filters, and so on.
## Prepare the database with any custom settings, and so on.
preImport()

## Run the CSDT import sequence
CSDT_Import()

## Run the test suite – sanity, performance, acceptance tests
runTestSuite()

## Report results to the team by email so they know about any failures first thing in the morning
runReports()
```

The script runs as a cron job at five past midnight every night.

37

Using the Developer Tools Command Line Interface (CLI) to Create Reusable Modules

With the Developer Tools kit you are able to reuse and share resources in the form of modules. Modules are workspaces that are not site-specific and contain resources such as Templates, flex families, and ElementCatalog entries. Unlike the standard export/import functionality where assets are added to sites using natural mappings, modules typically utilize site overriding so they can be imported into any site you designate.

Topics:

- Creating a Reusable Model

- List the Resources in the WebCenter Sites Instance

- List Start Menu Items

- Export All Resources to a Workspace

- Inspect the Module's Content

- Archive the Module

- Import the Module to a WebCenter Sites Instance

37.1 Creating a Reusable Model

Artie has a flex family with a flex definition that he wants to reuse in other sites. He also has a Template asset associated with the flex definition. In the following scenario, Artie creates a module containing these resources.

This scenario uses the command line interface to create a module containing the resources Artie and his team developed in Using Developer Tools to Manage and Exchange Resources.

Note:

To use the command line interface, Artie must specify the user name and password of a general administrator in each command he executes. This user must be a member of the `RestAdmin` group. In this scenario, Artie uses `fwadmin/xceladmin`.

See the following sections:

- List the Resources in the WebCenter Sites Instance

- List Start Menu Items

- [Export All Resources to a Workspace](#)

- [Inspect the Module's Content](#)

- [Archive the Module](#)

- [Import the Module to a WebCenter Sites Instance](#)

37.2 List the Resources in the WebCenter Sites Instance

Artie uses the command line interface to browse his WebCenter Sites instance. He uses the `resources=@ALL_ASSETS` and the `fromSites=Acceptance` selectors to list all the assets of the Acceptance site. The command Artie uses is `listcs`, which lists all the resources on his WebCenter Sites instance.

Command:

```
${sites-shared}/config/export/envision/cs_workspace$
java - Xbootclasspath/a:lib/servlet-api.jar -jar
developer-tools-command-line.jar http://<host>:<port>/
     <context>/ContentServer username=fwadmin password=xceladmin
   resources=@ALL_ASSETS fromSites=Acceptance cmd=listcs
```

Output:

```
Resource Type ||| Resource Id ||| Name ||| Description ||| Modified On
--------------------------------------------------------------------------------
Author_CD ||| 1297720502271 ||| fictionAuthor (status=ED) ||| authors who write fiction |||
2011-02-17 15:10:41
Author_A ||| 1297720502260 ||| authorName (status=ED) ||| author name ||| 2011-02-17 14:46:40
Author_A ||| 1297720502265 ||| authorBio (status=ED) ||| author biography ||| 2011-02-17 14:46:40
Author_A ||| 1297720502289 ||| 1297720502289 (status=VO) ||| author birthplace ||| 2011-02-17 15:12:35
Author_A ||| 1297720502293 ||| DOB (status=PL) ||| date of birth ||| 2011-02-17 14:46:40
Author_A ||| 1297720502305 ||| birthPlace (status=PL) ||| place of birth ||| 2011-02-17 15:10:22
Template ||| 1297720502331 ||| welcome (Typeless, status=ED) ||| welcome page ||| 2011-02-17 23:18:18
```

Artie notes that there are five `Author_A` flex attribute instances (one of which is voided), one `Author_CD` flex definition, and a Template asset.

37.3 List Start Menu Items

Artie further uses the command line interface to browse for any start menu items that are assigned to the **Acceptance** site.

Command:

```
${sites-shared}/config/export/envision/cs_workspace$
java - Xbootclasspath/a:lib/servlet-api.jar -jar
    developer-tools-command-line.jar http://<host>:<port>/
    <context>/ContentServer username=fwadmin password=xceladmin
  resources=@STARTMENU:* fromSites=Acceptance cmd=listcs
```

Output:

```
Resource Type ||| Resource Id ||| Name ||| Description ||| Modified On
--------------------------------------------------------------------------------
@STARTMENU ||| 1297720502206 ||| Find Author ||| null ||| -
@STARTMENU ||| 1297720502214 ||| Find Author Attribute ||| null ||| -
@STARTMENU ||| 1297720502222 ||| Find Author Definition ||| null ||| -
@STARTMENU ||| 1297720502230 ||| Find Author Filter ||| null ||| -
@STARTMENU ||| 1297720502238 ||| Find Author Parent ||| null ||| -
@STARTMENU ||| 1297720502246 ||| Find Author Parent Def ||| null ||| -
```

```
@STARTMENU ||| 1297720494070 ||| Find CSElement, FirstSiteII ||| Find CSElement ||| -
@STARTMENU ||| 1297720494086 ||| Find Page, FirstSiteII ||| Find Page ||| -
@STARTMENU ||| 1297720494078 ||| Find SiteEntry, FirstSiteII ||| Find SiteEntry ||| -
@STARTMENU ||| 1297720494066 ||| Find Template, FirstSiteII ||| Find Template ||| -
@STARTMENU ||| 1297720502210 ||| New Author ||| null ||| -
@STARTMENU ||| 1297720502218 ||| New Author Attribute ||| null ||| -
@STARTMENU ||| 1297720502226 ||| New Author Definition ||| null ||| -
@STARTMENU ||| 1297720502234 ||| New Author Filter ||| null ||| -
@STARTMENU ||| 1297720502242 ||| New Author Parent ||| null ||| -
@STARTMENU ||| 1297720502250 ||| New Author Parent Def ||| null ||| -
@STARTMENU ||| 1297720501427 ||| New CSElement ||| null ||| -
@STARTMENU ||| 1297720494052 ||| New Page, FirstSiteII ||| New Page ||| -
@STARTMENU ||| 1297720501431 ||| New SiteEntry ||| null ||| -
@STARTMENU ||| 1297720501435 ||| New Template ||| null ||| -
```

37.4 Export All Resources to a Workspace

Before he creates a module, Artie first exports all resources into a particular workspace.

Artie wants to create a module using all of the resources listed in List the Resources in the WebCenter Sites Instance and List Start Menu Items. He runs the following command to export all of the resources, at one time, into the specified workspace:

Command:

```
${sites-shared}/config/export/envision/cs_workspace$
java - Xbootclasspath/a:lib/servlet-api.jar -jar
  developer-tools-command-line.jar http://<host>:<port>/
  <context>/ContentServer username=fwadmin password=xceladmin
  resources=@STARTMENU:*;@ALL_ASSETS fromSites=Acceptance cmd=export
  datastore=authorModule
```

Output:

```
*** Exporting batch 1298385511005
Exporting ASSETDATA Author_CD:1297720502271 (batch 1298385511005)
Exporting ASSETDATA Author_A:1297720502260 (batch 1298385511005)
Exporting ASSET_TYPE Author_A (batch 1298385511005)
Exporting ELEMENTCATALOG OpenMarket/Xcelerate/AssetType/Author_A/LoadSiteTree (batch 1298385511005)
Exporting ELEMENTCATALOG OpenMarket/Xcelerate/AssetType/Author_A/AppendSelectDetails (batch
1298385511005)
Exporting ELEMENTCATALOG OpenMarket/Xcelerate/AssetType/Author_A/AppendSelectDetailsSE (batch
1298385511005)
Exporting ELEMENTCATALOG OpenMarket/Xcelerate/AssetType/Author_A/IndexAdd (batch 1298385511005)
Exporting ELEMENTCATALOG OpenMarket/Xcelerate/AssetType/Author_A/IndexReplace (batch 1298385511005)
Exporting ELEMENTCATALOG OpenMarket/Xcelerate/AssetType/Author_A/IndexCreateVerity (batch
1298385511005)
Exporting ELEMENTCATALOG OpenMarket/Xcelerate/AssetType/Author_A/ContentDetails (batch 1298385511005)
Exporting ELEMENTCATALOG OpenMarket/Xcelerate/AssetType/Author_A/ContentForm (batch 1298385511005)
Exporting ELEMENTCATALOG OpenMarket/Xcelerate/AssetType/Author_A/PostUpdate (batch 1298385511005)
Exporting ELEMENTCATALOG OpenMarket/Xcelerate/AssetType/Author_A/PreUpdate (batch 1298385511005)
...
```

All asset types for the flex family are included in the export. In addition, all elements belonging to those types are included as well. This information, although not usually modified, is necessary to make the module Artie is creating reusable on other WebCenter Sites instances.

37.5 Inspect the Module's Content

Artie inspects the authorModule workspace on his file system.

Figure 37-1   authorModule
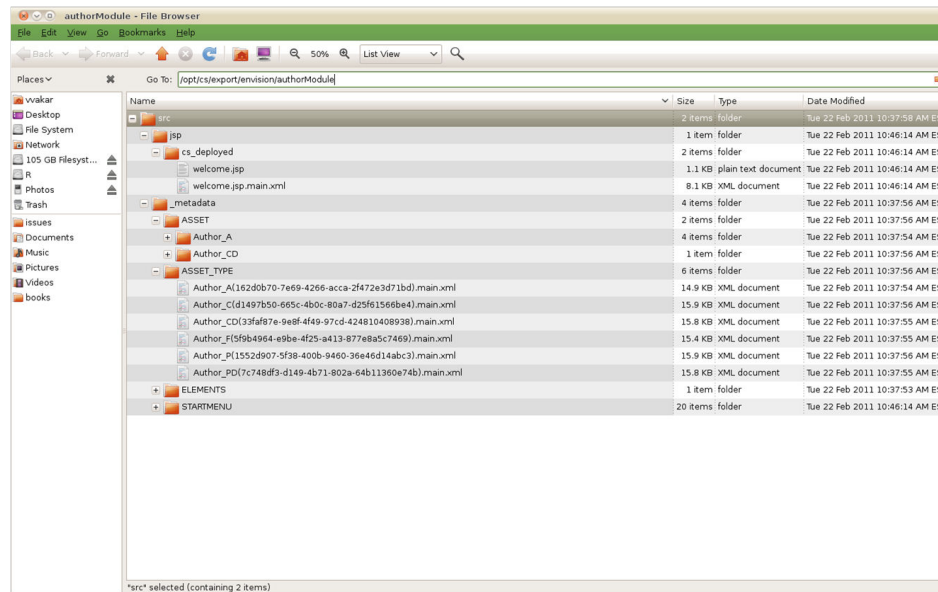
Artie notes that the Template asset, flex family members, asset types, and start menu items were all exported to the workspace on his file system.

37.6 Archive the Module

Artie is just one step away from importing the authorModule to a WebCenter Sites instance.

Artie creates a `.zip` file archive of the authorModule workspace and saves it.

37.7 Import the Module to a WebCenter Sites Instance

Artie is ready to import the module into a sample site.

1. Artie unzips the module into the workspace location of the target WebCenter Sites instance.

Figure 37-2   authorModule in Workspace Location
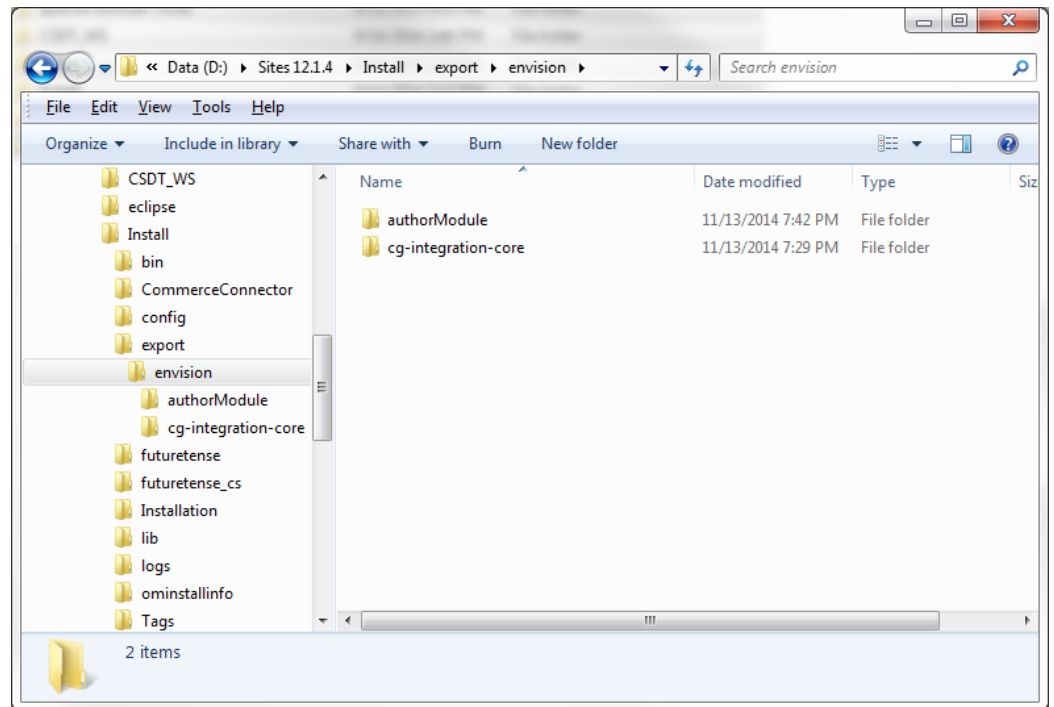
2. Using the command line interface, Artie imports the asset types and start menu items into the target WebCenter Sites instance.

Command:

```
${sites-shared}/config/JSKdemo/ContentServer>
    java - Xbootclasspath/a:lib/servlet-api.jar -jar
    developer-tools-command-line.jar http://<hots>:<port>/
    <context>/ContentServer username=fwadmin password=xceladmin
    resources=@ALL_NONASSETS cmd=import datastore=authorModule
    toSites=FirstSiteII
```

Output:

```
*** Importing batch 1298052933085
Importing DSKEY @STARTMENU-4340b65d-a9e4-4131-ac7f-51185a79b18d (batch 1298052933085)
Saved 1297720494070 (batch 1298052933085)
Importing DSKEY @STARTMENU-0a2decd4-b6be-418c-9992-a4332480bb20 (batch 1298052933085)
Saved 1297720501435 (batch 1298052933085)
Importing DSKEY @STARTMENU-66edea6d-218e-41b7-b5ac-ec3453bd53b7 (batch 1298052933085)
Saved 1297720502210 (batch 1298052933085)
Importing DSKEY @STARTMENU-c416c0d6-98a7-4ebf-babb-78d0699698de (batch 1298052933085)
Saved 1297720502230 (batch 1298052933085)
Importing DSKEY @ASSET_TYPE-162d0b70-7e69-4266-acca-2f472e3d71bd (batch 1298052933085)
Dependency @ASSET_TYPE-Author_A already exists, skipping.
Importing DSKEY @ELEMENTCATALOG-OpenMarket/Xcelerate/AssetType/Author_A/LoadSiteTree (batch
1298052933085)
Saved OpenMarket/Xcelerate/AssetType/Author_A/LoadSiteTree (batch 1298052933085)
Importing DSKEY @ELEMENTCATALOG-OpenMarket/Xcelerate/AssetType/Author_A/AppendSelectDetails (batch
1298052933085)
Saved OpenMarket/Xcelerate/AssetType/Author_A/AppendSelectDetails (batch 1298052933085)
Importing DSKEY @ELEMENTCATALOG-OpenMarket/Xcelerate/AssetType/Author_A/AppendSelectDetailsSE
(batch 1298052933085)
Saved OpenMarket/Xcelerate/AssetType/Author_A/AppendSelectDetailsSE (batch 1298052933085)
```

```
Importing DSKEY @ELEMENTCATALOG-OpenMarket/Xcelerate/AssetType/Author_A/IndexAdd (batch
1298052933085)
Saved OpenMarket/Xcelerate/AssetType/Author_A/IndexAdd (batch 1298052933085)
...
```

3. Artie opens the Admin interface for the FirstSiteII sample site and confirms that the asset types and start menu items were imported successfully.

Figure 37-3   Start Menus for FirstSiteII
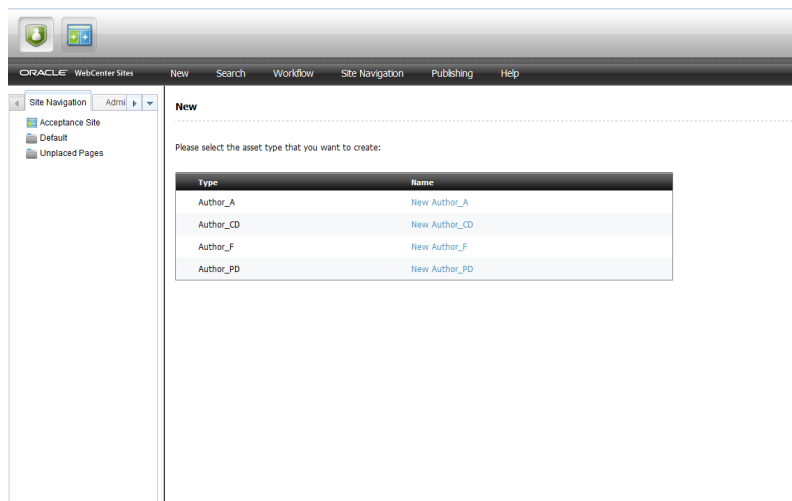
4. Now, Artie imports the assets.

Command:

```
${sites-shared}/config/JSKdemo/ContentServer>
java - Xbootclasspath/a:lib/servlet-api.jar -jar
  developer-tools-command-line.jar http://<host><:port>/
  <context>/ContentServer username=fwadmin
  password=xceladmin resources=@ALL_ASSETS cmd=import datastore=authorModule
  toSites=FirstSiteII
```

Output:

```
*** Importing batch 1298480206533
Importing DSKEY Author_A-cbf4d8aa-d23a-4f0d-b55d-a87a0e9bbf33 (batch 1298480206533)
Dependency @ASSET_TYPE-Author_A already exists, skipping.
Saved Author_A:1297837451977 (batch 1298480206533)
Importing DSKEY Author_A-42afd458-e90c-4e18-a4b6-47d322b46414 (batch 1298480206533)
Dependency @ASSET_TYPE-Author_A already exists, skipping.
Saved Author_A:1297837451981 (batch 1298480206533)
Importing DSKEY Author_A-9fe04c6e-36e7-4ee3-8c76-8c02edf74136 (batch 1298480206533)
Dependency @ASSET_TYPE-Author_A already exists, skipping.
Saved Author_A:1297837451985 (batch 1298480206533)
Importing DSKEY Author_CD-71d6067b-35f6-47f4-ae97-3876303abb37 (batch 1298480206533)
Importing DSKEY Author_A-ada2d6be-ef14-4766-b446-911bfa838835 (batch 1298480206533)
Dependency @ASSET_TYPE-Author_A already exists, skipping.
Saved Author_A:1297837451989 (batch 1298480206533)
Dependency @ASSET_TYPE-Author_C already exists, skipping.
Dependency @ASSET_TYPE-Author_P already exists, skipping.
Dependency @ASSET_TYPE-Author_CD already exists, skipping.
Dependency @ASSET_TYPE-Author_PD already exists, skipping.
Dependency @ASSET_TYPE-Author_F already exists, skipping.
Dependency @ASSET_TYPE-Author_A already exists, skipping.
Saved Author_CD:1297837451993 (batch 1298480206533)
```

```
Importing DSKEY Template-89b05c0f-227b-4dcb-961e-2ab6e6af2dae (batch 1298480206533)
Saved Template:1297837452000 (batch 1298480206533)
*** Completed importing batch 1298480206533
```

5. Artie verifies that the flex definition is imported into the FirstSiteII sample site successfully.

Figure 37-4   Inspect Author Definition
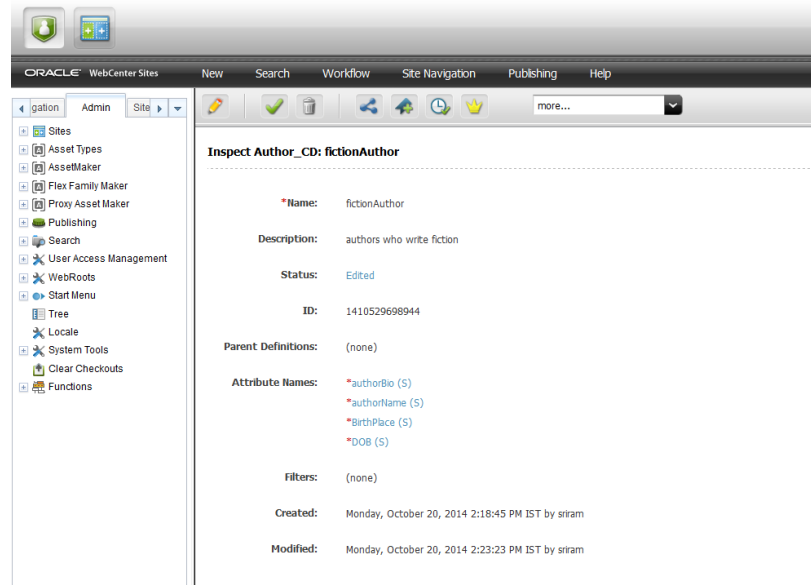
6. Using the command line interface, he also imports the Template asset. He then opens the Admin interface again to verify the Template asset is imported correctly.

Figure 37-5   Template Welcome
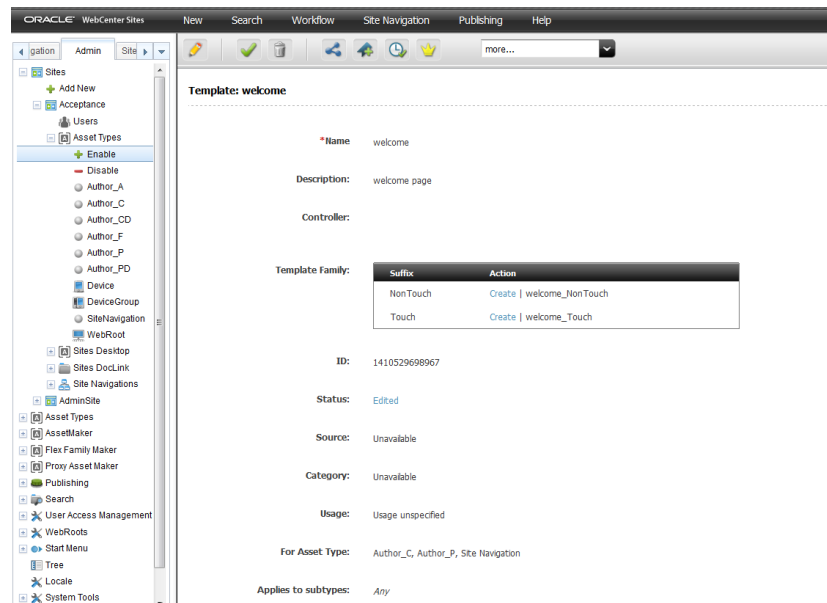
The entire module is imported successfully into the FirstSiteII sample site. This module can be reused and imported into any WebCenter Sites instance.

Part VI

Managing Caching

To manage caching efficiently, you need to know what resultset caching is and how you can accurately cache resultsets and flush them from the cache. Other topics that will help you are rendering engine cache and its components, enabling CacheManager to clear all caches, and understanding how rendered object caching works in the WebCenter Sites platform.

Topics:

- Understanding Page Design and Caching

- Working with Resultset Caching and Queries

- Using Cache Management with WebCenter Sites

- Using Advanced Page Caching Techniques

38

Understanding Page Design and Caching

Caching your web pages can improve your site's performance. Whether your site is static or dynamic, you should design your site so that part or all of a given page is cached.

Topics:

- About Modular Page Design

- About Caching

- Double-Buffered Caching

38.1 About Modular Page Design

A modular page design is composed of multiple elements. It has several benefits such as improved system performance and reusability of common design elements.

- Improved system performance, allowing you to develop an efficient caching strategy.

- Common design elements, like navigation bars, may be coded once and used on multiple web pages.

This figure shows a simple modular page:

Figure 38-1   Modular Page
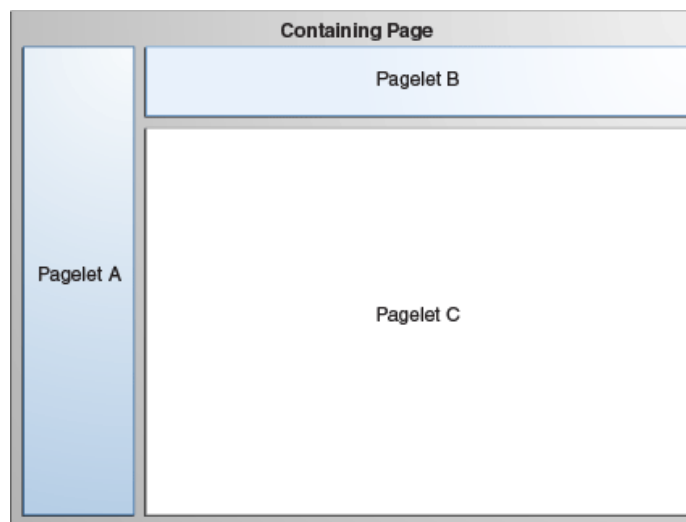
Each rectangle represents the generated output of one or more elements, referred to as a pagelet. These pagelets are called by a containing page, which allows you to do the following:

- Lay out how the pagelets appear on the finished page.

- Define code that must be evaluated each time the page is viewed, such as custom Access Control List (ACL) checking code.

This strategy lets you code an element once and use it in many places in your website.

38.2 About Caching

WebCenter Sites allows you to cache entire web pages and the components that make up those web pages. An efficient page caching strategy helps you improve system performance by reducing load.

Two members of the WebCenter Sites product family implement page caching:

- WebCenter Sites caches pages on the WebCenter Sites system.

- Satellite Server provides a second level of caching for your WebCenter Sites system, and can also be used as a remote cache for your web pages.

WebCenter Sites uses both the WebCenter Sites and Satellite Server caches to create an efficient caching strategy.

38.2.1 WebCenter Sites Caching

It speeds processing when pagelets generated by requests to the ContentServer servlet can be cached on disk. If a page is accessed frequently and its content depends on a small number of parameters, then it is a good candidate for disk caching.

To disk-cache a pagelet, use one of the following tags:

Table 38-1   Caching Tags

JSP Tag	XML Tag
satellite:page	SATELLITE.PAGE
render:satellitepage	RENDER.SATELLITEPAGE

If the pagelet that you want to cache is not in the disk cache already, ContentServer adds it to the cache and then serves the pagelet.

38.2.2 BlobServer and Caching

The term *blob* is an acronym for Binary Large OBject. Although a blob is usually an image file, a blob can be any binary object, including a Microsoft Word file or a spreadsheet. Most websites serve several blobs.

WebCenter Sites offers a special servlet called BlobServer. The BlobServer gathers a blob from a table and performs all relevant security checks.

You can access BlobServer with the BlobServer tags:

- satellite:blob

- render:satelliteblob

Both of these tags cache blobs in the WebCenter Sites and Satellite Server caches. See *Oracle Fusion Middleware Tag Reference for Oracle WebCenter Sites Reference*.

Deleting Blobs from the WebCenter Sites Memory Cache

To delete a specific blob from the WebCenter Sites cache, you must edit the BlobServer URL:

- Rename the `blobtable` parameter to `flushblobtable`.

- Authenticate as a user with `SiteGod` privileges by passing credentials through the `authusername` and `password` parameters.

For example:

```
http://hostname:port/servlet/BlobServer?blobcol=urlpicture&blobheader=image
%2Fgif&blobkey=id&flushblobtable=NewPortalImage&blobwhere=22&authusername=username&au
thpassword=password
```

To delete all blobs, rename the `blobtable` parameter to `flushblobtables` (notice the "s") and set it to `true`.

38.2.3 Satellite Server Caching

Satellite Server is automatically installed with WebCenter Sites, and provides an additional layer of caching. To improve your WebCenter Sites system's performance, you can add remote Satellite Server systems, placing your content closer to its intended audience.

Satellite Server caches pages, pagelets, and blobs to disk or to memory. You can use the `Inventory` servlet to view the contents of the memory and disk caches in varying degrees of detail. Note that items cached on Satellite Server are not protected by WebCenter Sites APIs. You can overcome this limitation by using the caching strategy outlined in Pagelet Caching Strategies.

Satellite Server caches small items to memory and large items to disk. You control the definitions of small and large through the `file_size` property. See Managing Satellite Server JSON File Properties in *Oracle Fusion Middleware Property Files Reference for Oracle WebCenter Sites*.

On a busy site, each Satellite Server system's cache fills up quickly with the most popular pages. When the cache is full, Satellite Server deletes old pages to make room for new ones. Satellite Server uses a Least Recently Used algorithm (LRU) to determine which items should be removed from the cache. In other words, when a new page needs to be cached, Satellite Server removes the page that hasn't been accessed for the longest time.

38.2.3.1 Cache Expiration

Page and pagelet expiration on Satellite Server is specified in the `sscacheinfo` column of the `SiteCatalog` table. Each time a page or pagelet is invoked through Satellite Server, Satellite Server processes the **sscacheinfo** field's value and determines when the page or pagelet should expire. See CacheInfo String Syntax in Using Advanced Page Caching Techniques.

> **Note:**
>
> It is possible to override the `sscachinfo` expiration information for pagelets by specifying the `cachecontrol` attribute in the `satellite.page` and `render.satellitepage` tags. However, this practice is discouraged because it can lead to unpredictable behavior. Some pagelets may be accessed through the default method (without the `cachecontrol` attribute), while others may be accessed with an override. The first method invoked will set the expiration for Satellite Server, and the second one will have no effect on the expiration.

Blobs cached on Satellite Server expire according to the following algorithm:

- You can use Satellite Server tags to override the default expiration time on a blob-by-blob basis.

- If there is no Satellite tag to override the default expiration, Satellite Server gets the expiration time from the value of the `satellite.blob.cachecontrol.default` property. This property is described in Page Caching Properties.

- If no value is set for the `satellite.blob.cachecontrol.default` property, Satellite Server gets the expiration time from the value of the `expiration` property, described in Satellite Server Properties.

38.2.3.2 Caching with the Satellite Servlet

This topic describes how the Satellite servlet caches web pages and how to implement caching with the Satellite servlet in tandem with modular page design to create a fast, efficient website.

How the Satellite Servlet Caches Pages

The Satellite servlet allows caching at the pagelet level. To implement caching with the Satellite servlet, use Satellite Server XML or JSP tags in your WebCenter Sites pages, and access pages using special Satellite URLs.

For example, suppose that you used the Satellite servlet to implement pagelet-level caching on a web page named `myPage`. This figure shows that this page is composed of a containing page and three pagelets; A, B, and C. The containing page and pagelets A and B are cached on a Satellite Server system, but pagelet C is not cached:

Figure 38-2   Web Page Named myPage
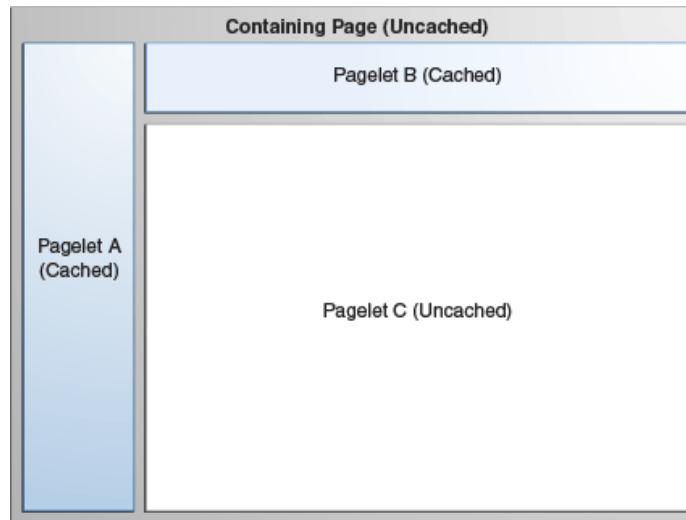
The following occurs when a user requests `myPage`:

1. Satellite Server examines the URL. If it is a Satellite URL, the Satellite servlet gets the cached copy of the containing page. The servlet then looks for pointers to pagelets that are not currently in its cache, and requests those pagelets from WebCenter Sites. So, in our example, the Satellite servlet gets the containing page, and gets Pagelets A and B from its cache.

2. The Satellite servlet requests Pagelet C from WebCenter Sites.

3. WebCenter Sites parses the appropriate XML to create Pagelet C and sends it to the Satellite servlet.

4. The Satellite servlet assembles Pagelets A, B, and C into the page, and sends the assembled page to the requester. The servlet also caches Pagelet C.

Implementing Caching with the Satellite Servlet

To implement pagelet-level caching with the Satellite servlet, add Satellite tags to your WebCenter Sites templates. You do not develop any XML, JSP, or Java code on Satellite Server systems. In fact, Satellite Server does not know how to parse XML.

The Satellite tags in your elements are interpreted by the Java code you installed as part of Satellite Server. If this code is being called with a Satellite URL, it generates the information that the Satellite servlet uses to cache and construct the pagelets. If you do not call an element containing Satellite tags with a Satellite URL, the resulting page functions as if the Satellite tags were WebCenter Sites tags.

Satellite URLs look like the following example:

`http://host:port/servlet/Satellite?pagename=page`

where `host` and `port` are the host name and port number of your Satellite Server computer, and `page` is the name of the page you are requesting. A Satellite URL can also include name/value pairs you want to pass to the called page.

Caching a Pagelet

The following sample code uses the `render:satellitepage` tag to call a pagelet. If the pagelet is not in Satellite Server's cache, the Satellite servlet loads and caches the page. If the pagelet encounters an error during the processing and cannot be evaluated, it is not cached.

The `render:satellitepage` tag (and the `satellite:page` tag and their XML equivalents) identifies a cached pagelet by the pagename and name/value pairs passed to it. If the parameters or the name/value pairs differ from one invocation to another, a different pagelet is cached, even if the content generated is the same. It is important to use name/value pairs to pass arguments to a pagelet through these tags.

Values passed through the ICS object pool, ICS List pool, page attribute context, and session (including session variables) may not be available to all called pagelets. Nested pagelets are not always be called at the same time as the parent. Furthermore, pagelets that rely on session or context data are rarely cacheable anyway, so attempting to cache them can result in unexpected behavior.

All parameters passed to a nested pagelet through `render:satellitepage` (and the `satellite:page` tag, and their XML equivalents) must be specified in the SiteCatalog as page criteria. This determines which parameters are relevant when building a pagelet for caching. Parameters other than those listed in the SiteCatalog are not permitted (an error indicating this will be written to the log).

```
<cs:ftcs>
<html>
  <body>
    <render:satellitepage pagename="My/Sample/Page" />
  </body>
</html>
</cs:ftcs>
```

Caching a Blob

Using Satellite tags to load and cache a blob is similar to the way you use Satellite tags to load and cache a pagelet. The following sample code adds to the previous example by calling a blob and a pagelet.

```
<html>
<body>
<!-- NOTE: This will fail if list has no content (== null) -->

<ics:setvar name="category" value="logo"/>
<ics:setvar name="errno" VALUE="0"/>
<ics:selectto from="SmokeImage" list="imagelist" where="category" limit="1"/>

<ics:then>
<!-- Test a blob -->

<render:satelliteblob service="img src"blobtable="SmokeImage" blobkey="id"
blobwhere="imagelist.id" blobcol="urlpicture" blobheader="image/gif"cachecontrol="*:
30:0 */*/*"alt="imagelist.alttext" border="0" />
</ics:then>

<render:satellitepage pagename="QA/Satellite/Functional/
xml/"pagelet1"cachecontrol="never"/>
</body>
</html>
```

The following actions are defined in the above code:

- The `ics:selectto` tag performs a simple SQL query that retrieves a blob from the database. Results are returned in the form of an `IList` named `imagelist`.

- The `satellite:blob` tag loads the blob that was retrieved from the database. As with the `satellite.page` tag, if the blob is not in Satellite's cache, Satellite

loads and caches the blob. The `cachecontrol` parameter is set so that the blob expires at a given time; in this case, every 30 minutes.

Never-Expiring Blobs

If there are binary files (or blobs) on your site that seldom change or never change, such as company logos, and you are using the Satellite servlet to cache at the pagelet level, you can improve performance by using an alternative method to serve these blobs.

To serve never-expiring blobs:

1. Copy the never-expiring images to all your Satellite Server hosts. Place them under the doc root for your web server.

2. Access the images through `<img src="pathname">` HTML tags rather than through `satellite:blob` Satellite tags.

For example, consider a never-expiring corporate logo file named `CorporateLogo.gif`. To use the alternative method of serving blobs, you would first copy the file to the web server's doc root on all your Satellite Server hosts. Then, instead of serving this logo through a `satellite.blob` tag, your element could simply use a tag like the following:

```
<img src="CorporateLogo.gif">
```

> **Note:**
>
> Be careful when using this mechanism for serving never-expiring images. For example, Satellite Server cannot warn you that one of the Satellite Server hosts does not contain the same image file as the other hosts.

```
http://myloadbalancer:1234/servlet/ContentServer?pagename=myPage
```

The expiration of the page is controlled by the `expiration` property. See *Oracle Fusion Middleware Property Files Reference for Oracle WebCenter Sites*.

38.2.3.3 Viewing the Contents of the Satellite Server Cache

The Inventory servlet lets you view the various items stored in the cache. You invoke the Inventory servlet by using the following URL:

```
http://host:port/servlet/Inventory?
username=username&password=passwordword&detail=value
```

where parameters are as defined in this table:

Table 38-2   Inventory Servlet Parameters

Parameter	Description
`host:port (required)`	The host name and port number of the Satellite Server host whose cache you want to view.
`username (required)`	The user name that you enter to log you in to the Satellite Server host.

Table 38-2   (Cont.) Inventory Servlet Parameters

Parameter	Description
`password (required)`	The password that you enter to log you in to the Satellite Server host.
`detail (optional)`	The type of information you want the Inventory servlet to display. Valid values are: • `names`: Displays the header information, plus the page names of the pages in the cache. • `keys`: Displays the header information, plus the page names and keys of the items in the cache. If you do not supply the `detail` parameter, or if you set its value to be anything other than `name` or `keys`, the header information displays.

The header contains the information types defined in this table:

Table 38-3   Information Types

Information type	Description
Remote host	The host that this Satellite Server system forwards requests to.
Maximum cache objects	The maximum number of items allowed in the cache.
Current size	The number of items currently in the cache.
Cache check interval	How often the cache is checked for expired items, in minutes.
Default cache expiration	The value of the `expiration` property.
Minimum file size (in bytes)	Items larger than this value are stored in files. Items smaller than this value are stored in RAM.

38.2.3.3.1 CacheManager

The CacheManager object maintains both the WebCenter Sites and Satellite Server caches. CacheManager can perform the following functions:

- Log pagelets in the cache tracking tables.

- Keep a record of the content (assets) that pages and pagelets contain by recording cache dependency items in cache-tracking tables. Cache dependency items are items that, when changed, invalidate the cached pages and pagelets that contain them. A cache dependency item is logged as a dependency for the current page and all parent pages.

- Remove pages and pagelets containing invalid items from the WebCenter Sites and Satellite Server caches.

- Rebuild the WebCenter Sites and Satellite Server caches with updated pages and pagelets after the invalid pages have been removed.

See Using Cache Management with WebCenter Sites.

38.2.3.3.2 The SiteCatalog Table

The WebCenter Sites `SiteCatalog` table lists the pages and pagelets generated by WebCenter Sites. An element must have an entry in the `SiteCatalog` table to be cached on WebCenter Sites and Satellite Server.

The fields in the `SiteCatalog` table set the default behavior of a WebCenter Sites page, including default caching behavior. See Creating Template Assets and Creating SiteEntry Assets.

38.2.3.3.3 The Cache Key

Items stored in the WebCenter Sites and Satellite Server caches are assigned a *cache key*. The cache key uniquely identifies each item in the cache. CacheManager locates items in the cache using the cache key. WebCenter Sites and Satellite Server generate cache keys automatically, based on the values in the **pagename**, **resargs**, and **pagecriteria** fields of the `SiteCatalog` table, and other internal data.

38.2.3.3.3.1 pagecriteria and the Cache Key

You can include variables used by the page in the cache key by specifying them in a comma-separated list in the **pagecriteria** field of the `SiteCatalog` table. For example, suppose that you have a page called `myPage` that uses the values `red` and `blue`.

To include `red` and `blue` in the `myPage` cache key, enter the following:

- `favoritecolor,second_favoritecolor` in the **pagecriteria** column

- `favoritecolor=red&second_favoritecolor=blue` in the **resargsl** column

WebCenter Sites and Satellite Server use the `pagecriteria` and parameters that are passed to cached pages to help generate the cache keys. If the parameters differ from one invocation to another, a different page is cached even if the content being generated is the same. For example:

```
http://mysatellite:1234/servlet/ContentServer?
pagename=myPage&favoritecolor=red
```

calls a different page than:

```
http://mysatellite:1234/servlet/ContentServer?
pagename=myPage&second_favoritecolor=blue
```

whether or not the content being generated is the same. Values passed by the URL override values set in `pagecriteria`. For example, you have the `myPage` `pagecriteria` set to `red,blue`:

- If the URL passes a value of `green`, then `green,blue` (not `red,blue`) will go into `myPage's` cache key.

- If the URL passes values of `green,violet`, then `green,violet` (not `red,blue`) will go into `myPage's` cache key.

- If the URL passes values of `green,violet,yellow`, an error results.

If a page does not have `pagecriteria` set, the values in the **resargs** fields go into the cache key. As with `pagecriteria`, values passed by a URL override values specified in the **resargs** fields.

38.2.3.3.4 Caching Properties

The default cache settings for WebCenter Sites and Satellite Server are contained in the `wcs_properties.json` file. These properties can be modified using the Property Management Tool. See the *Oracle Fusion Middleware Property Files Reference for Oracle WebCenter Sites*.

Additional Satellite Server properties are contained in the `wcs_properties.json` file for remote Satellite Servers, located under the `config` directory, and must be modified manually.

38.2.3.3.5 Page Caching Properties

The following properties in `wcs_properties.json` control disk caching:

- `cs.freezeCache`: Controls whether the cache pruning thread should run to remove expired entries from the cache.

- `cs.nocache`: Disables the entire page cache.

38.2.3.3.6 Satellite Server Properties

Satellite Server has two sets of properties in the `wcs_properties.json` file (see the *Oracle Fusion Middleware Property Files Reference for Oracle WebCenter Sites*).

- One property in the `wcs_properties.json` file is categorized under Cache:

  `satellite.blob.cachecontrol.default`, which specifies a default value for the `cachecontrol` parameter for the `satellite.blob`, and `RENDER.SATELLITEBLOB` tags and their JSP equivalents.

- The other properties in the `wcs_properties.json` file are categorized under Satellite:

  - `cache_folder`: Specifies the directory into which Satellite Server caches pagelets to disk.

  - `file_size`: Separates disk-cached pagelets and blobs from memory-cached pagelets and blobs, according to size.

  - `expiration`: Sets the default value for the length of time blobs stay in Satellite Server's cache.

  - `cache_check_interval`: Controls the frequency of the cache cleaner thread, and therefore defines when expired objects are pruned from cache.

  - `cache_max`: Specifies the maximum number of objects (pagelets and blobs) that can be cached in both memory cache and disk cache combined at a time.

38.3 Double-Buffered Caching

WebCenter Sites and Oracle WebCenter Sites: Engage implement a double-buffered caching strategy. This caching uses the WebCenter Sites and Satellite Server caches in tandem. The double-buffered caching strategy ensures that pages are always kept in cache.

You can implement a similar caching strategy by using the CacheManager Java API. This only applies if you are if you are running the WebCenter Sites core and Satellite Server without any of the other CS modules or products. For more information about

the CacheManager Java API, see *Oracle Fusion Middleware Java API Reference for Oracle WebCenter Sites*.

Both the WebCenter Sites core and Satellite Server caches are maintained by the WebCenter Sites CacheManager object. CacheManager tracks when content changes by logging elements and the assets that those elements call in cache tracking tables.

When assets are updated and published, WebCenter Sites and Satellite Server caches are automatically flushed and updated in the order shown in this figure:

Figure 38-3   Order of Cache Flushing 1
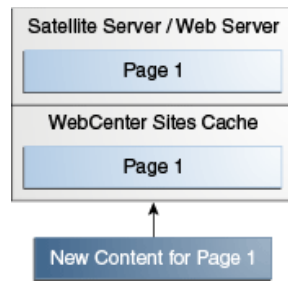
- In the following figure, content providers publish updated assets to the delivery system. CacheManager checks the cache tracking tables to see which cached items are affected by the updated assets.

Figure 38-4   Order of Cache Flushing 2
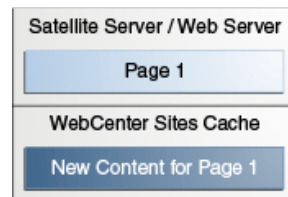
- In the following figure, CacheManager flushes the outdated `Page1` from the WebCenter Sites cache, then reloads the WebCenter Sites cache with the updated `Page1`. Any requests for `Page1` will be served the old version of `Page1` from the Satellite Server cache. This protects the WebCenter Sites computer from undue load as it deletes and rebuilds its cache.

Figure 38-5   Order of Cache Flushing 3
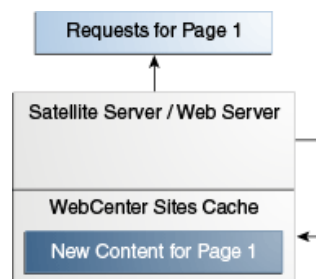
- In the following figure, CacheManager flushes the outdated items from the Satellite Server cache. As visitors come to the website and request `Page1`, the Satellite Server searches to see if `Page1` is in its cache. Because `Page1` is not in the Satellite Server cache, the request is passed on to WebCenter Sites.

Figure 38-6   Order of Cache Flushing 4
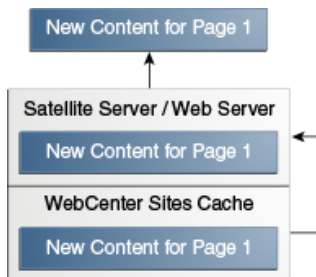
- The Satellite Server system's cache is filled with an updated version of `Page1`, taken from the WebCenter Sites cache. The updated page is served to the requestors. If `Page1` is requested again, the page is served from the Satellite Server cache.

38.3.1 About Implementing Double-Buffered Caching

The first step in implementing double-buffered caching on your website is to design modular pages, as described in About Modular Page Design. Once you have developed a modular page design, you implement a double-buffered caching strategy in three steps:

- Develop a pagelet caching strategy.

- Set how individual pages and pagelets are cached by using the **pagecriteria** field of the `SiteCatalog` table.

- Code your elements with Satellite tags.

38.3.1.1 Pagelet Caching Strategies

With a modular page design, caching occurs at the pagelet level. The containing page is never cached, so that any cached pagelets are always protected by ACLs. You choose which pagelets get cached based on how frequently they are updated.

This table summarizes the guidelines for caching pagelets:

Table 38-4   Guidelines for Caching Pagelets

Cache a Pagelet	Don't Cache a Pagelet
- If the content seldom changes. - If the pagelet does not contain logic that requires evaluation to work.	- If the content changes frequently. - If the content must be real time. - If the pagelet contains code that checks for ACLs, or other logic that requires evaluation to work.

An example of a modular page is shown in this figure:

Figure 38-7   Modular Page
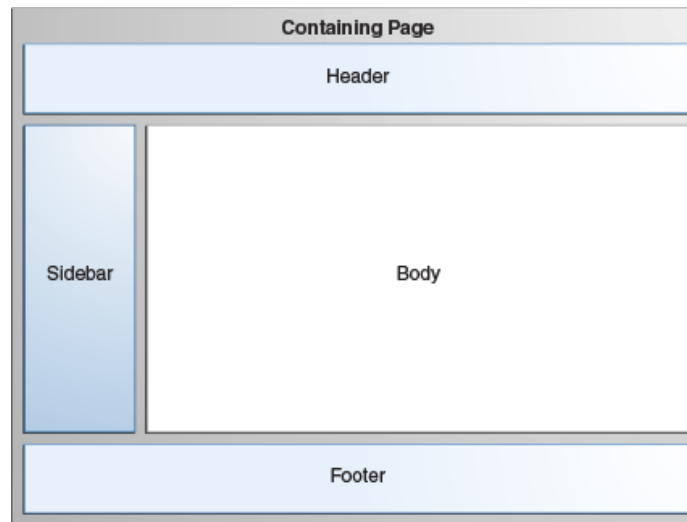
The containing page should never be cached. You can put logic that requires evaluation by WebCenter Sites into your pages, while still gaining the performance benefits of caching. Your page can also be protected by WebCenter Sites ACLs.

The header and footer pagelets in this example should be disk cached. They rarely get updated, and should be designed accordingly. The header and footer may be static HTML written into your template, or disk-cached content from WebCenter Sites.

The sidebar is also a good candidate for disk caching. It has a small number of variations, and its content is determined by a small number of parameters.

Determining how to cache the body pagelet is more complex. The contents of the body pagelet may depend on where the website visitor is in the site. There are three possible types of content for the body pagelet:

- The results of a search that the website visitor runs

- The results of a frequently run query

- An article

Your caching strategy should be consider the following points:

- If the content of the body pagelet is the result of a search based on parameters that the website visitor enters, you do not want to cache it. Such pages change for each visitor, and there is little benefit to caching them.

- If the content is the product of a standard query that visitors often use, you should use resultset caching. Caching frequently run queries in the memory cache improves performance. See Working with Resultset Caching and Queries.

- If the content of the body pagelet is the text of an article, you should cache the pagelet to disk.

38.3.2 Setting cscacheinfo

The values in the **cscacheinfo** field of the `SiteCatalog` table allow you to control how pages are cached on WebCenter Sites.

- You can change these properties for each page and pagelet in your website. For example, if you want a containing page element to be uncached on WebCenter Sites, set the values in **cscacheinfo** to `false`.

For more information about the **cscacheinfo** field, see Creating Template Assets.

38.3.3 Coding for Caching

- To implement double-buffered caching, code your elements with Satellite Server tags. If you are running WebCenter Sites and Satellite Server only, use the Satellite tags documented in the Satellite Server sections of the *Oracle Fusion Middleware Tag Reference for Oracle WebCenter Sites Reference*.

Automatic cache maintenance depends on you logging your assets in the cache tracking tables. If you use the `ASSET.LOAD` tag to load an asset, that asset is automatically logged in the cache tracking tables. For those sections where `ASSET.LOAD` is not used, use the `RENDER.LOGDEP` tag to log content in the cache tracking tables.

> **Note:**
>
> Cache dependencies are logged only if a page or pagelet is cached on WebCenter Sites. If a page is uncached on WebCenter Sites but cached on Satellite Server, that page is not automatically flushed from the cache when its content is updated.

38.3.4 Caching and Security

Cached pagelets require special security considerations as you design your site and develop your caching strategy. The following sections outline security considerations for pages cached in the WebCenter Sites and Satellite Server caches:

- WebCenter Sites Security

- Satellite Server Security

38.3.4.1 WebCenter Sites Security

Pagelets that are disk cached on WebCenter Sites are bound by the WebCenter Sites ACLs, allowing you to use those ACLs to prevent unauthorized access to a page.

Note, however, that although WebCenter Sites checks the ACL of a containing page, it does not check the ACLs of the pagelets that the containing page calls. For example, suppose that your site uses three ACLs: `Open`, `Secret`, and `TopSecret`. Your containing page can be viewed by members of the `Open` ACL, but it calls pagelets that should be viewed only by members of the `Secret` and `TopSecret` ACLs. Because WebCenter Sites only checks a visitor's ACL of the containing page, visitors with the `Open` ACL can view content meant for members of the `Secret` and `TopSecret` ACLs.

To ensure that all the relevant ACLs are checked:

1. Include the ACL for the page that you want to protect in that page's cache criteria, as shown in the following sample code:

```
<render.satellitepage pagename="innerwrapper" userAcl="SessionVariables.member"
c="Article" cid="123">
```

2. In the pagelet, insert code to check the ACLs, as shown in the following sample:

```
<asset.load name="art" type="Variables.c" OBJECTID="Variables.cid"/>
<ASSET.GET NAME="art" FIELD="myACL"/> <!-- note you need a column in your db to
support this -->
<IF COND="Variables.userACL=Variables.myACL">
<THEN>
<render.satellitepage pagename="protected_art_tmpl1" c="Variables.c"
cid="Variables.cid"/>
</THEN>
<ELSE>
<render.satellitepage pagename="accessDenied"/>
</ELSE>
</IF>
```

38.3.4.2 Satellite Server Security

Pagelets to be cached on Satellite Server are only bound by WebCenter Sites ACLs under the following circumstances:

• If they are retrieved from the WebCenter Sites cache.

• If they must be generated by WebCenter Sites to fulfill the page request.

If a pagelet is served from the Satellite Server cache, it is no longer protected by WebCenter Sites ACLs.

To ensure that the content of your Satellite Server pages is secure, never cache your containing page and be sure that you put an ACL checking mechanism in the uncached container.

If your elements are coded with Satellite tags but you do not yet have Satellite Server installed, the page design considerations outlined in WebCenter Sites Security apply to you. Once Satellite Server is installed, however, WebCenter Sites checks the ACLs of uncached pagelets called from a containing page. The ACLs of pagelets cached on Satellite Server are not checked.

39

Working with Resultset Caching and Queries

Resultset caching is another means of improving the performance of your system. You can create queries that allow CatalogManager to cache resultsets accurately and flush those resultsets from the cache.

Topics:

39.1 About Resultset Caching and Queries

The resultset cache is maintained by the CatalogManager servlet. You or your system administrators set up resultset caching on all three systems (development, management, and delivery). Whenever the database is queried, WebCenter Sites serves a resultset, either cached or uncached. Resultset caching reduces the load on your database and improves the response time for queries.

The `wcs_properties.json` file provides global properties that set the size and timeout periods for all resultsets. You can add table-specific properties to the `wcs_properties.json` file that override the default settings on a table-by-table basis. These custom properties enable you to fine-tune your systems for peak performance.

Resultset caching reduces the load on your database and improves the response time for queries. Be sure to take the following steps:

- Set the default resultset caching properties in the `wcs_properties.json` file to values that make sense on each of your systems (development, management, testing, and delivery).

- Add table-specific resultset caching properties to the `wcs_properties.json` file to fine-tune the performance of all of your systems (development, management, testing, and delivery).

- Provide the correct table name for all of your queries so the resultsets are cached correctly and can be flushed correctly.

39.2 Caching Frameworks

By default, WebCenter Sites stores resultsets in the inCache framework. You have the option to switch to caching in hash table.

See Switching Between Caching Frameworks.

When resultset caching over inCache is enabled, the **System Tools** node (on the **Admin** tab of the Admin interface) displays the resultset over inCache tool, which provides statistical information about resultset caches and their contents. Note that resultset caching over inCache functions independently of page and asset caching over inCache. For more information about the inCache framework, its caching models, and system tools, see Working with Cache Management - Resultset Cache in *Oracle Fusion Middleware Administering Oracle WebCenter Sites*.

39.3 Database Queries

There are several ways you can query the WebCenter Sites database for information. See some examples here.

- Use the `ics.SelectTo` Java method, `SELECTTO` XML tag, or `ics:selectto` JSP tag.

- Use the `selectrow` command of the `ics.CatalogManager` Java method, the `CATALOGMANAGER` XML tag, and the `ics:catalogmanager` JSP tag.

- Use the `ics.SQL` Java method, `EXECSQL` XML tag, or `ics:sql` JSP tag.

- Use the `ics.CallSQL` Java method, `CALLSQL` XML tag, or `ics:callsql` JSP tag.

- Use the Search forms in the WebCenter Sites interface.

- Use a query asset.

- Use a `SEARCHSTATE` XML or JSP tag (flex assets only).

39.4 How Resultset Caching Works

When you query the database, the resultset from the query is cached—if resultset caching is enabled. The resultset cache is either a hash table or the inCache framework, based on how you configured the `rsCacheOverInCache` property in the `wcs_properties.json` file.

Table-specific properties override the default properties and enable you to fine-tune your systems for peak performance.

Default properties are resultset caching properties in the `wcs_properties.json` file that are assigned to all tables.

These properties are used for querying the database. If someone runs the same query and the data in the table remains unchanged since the last time the query was run, WebCenter Sites serves the information from the resultset cache rather than querying

the database again. Serving a resultset from cache is always faster than performing another database lookup.

The resultsets are organized by the name of the table that was associated with the query that generated the resultset. In other words, resultsets are cached against a table name.

Each time a table is updated (from either the WebCenter Sites interface or through a CatalogManager command in your custom elements), all the resultsets in the cache for that table are flushed. Resultsets are cached in the context of a single Java VM. Although Java VMs do not share resultsets, WebCenter Sites sends a signal to all the Java VMs in a cluster to flush the resultsets when they become invalid, while the synchronization feature has been enabled on all servers in the cluster.

39.5 Reducing the Load on the Database

Resultset caching reduces the load on your database in many ways. For example, at the time of serving it doesn't need a database connection, it's enabled only when there are uncached resultsets, etc.

- Serving a cached resultset does not open a database connection. WebCenter Sites attempts to obtain a resultset from the cache before it contacts the database. If the correct resultset exists, no contact is made with the database.

- When resultset caching is enabled but the appropriate resultset is not cached, WebCenter Sites obtains the resultset, stores it in the cache as an object, and then releases the database connection.

- When resultset caching is not enabled, WebCenter Sites cannot close the database connection until either the online page is completely rendered or the uncached resultset is explicitly flushed from the scope with a `flush` tag. When this occurs, your available database connections can be quickly used up (even on a relatively simple page).

As a general rule, resultset caching should be enabled for all of your database tables. Although there are times when you might have to limit either the number of resultsets that are cached or the length of time that they are cached for, it is rarely a good idea to disable resultset caching altogether.

> **Note:**
>
> Never disable resultset caching on the `ElementCatalog` table. If you do, the performance of your system will suffer greatly, especially if you are using JSP in any of your elements.

39.6 Specifying the Table Name

Always remember to associate a table name with a query to cache the resultset against that table. Then, whenever you update the table through the WebCenter Sites interface or through your own custom elements, CatalogManager flushes all the resultsets associated with that table.

The way that the table name is specified for a resultset depends on the type of query you are running. The following sections describe the most commonly used methods for querying the database and how you specify the table name for such a query.

- SELECTTO

- EXECSQL

- CALLSQL

- Search Forms in the WebCenter Sites Interface

- Query Asset

- SEARCHSTATE

39.6.1 SELECTTO

When you use the `ics.SelectTo` Java method, `SELECTTO` XML tag, or `ics:selectto` JSP tag, you must specify the name of the table with a `FROM` parameter (clause). For example:

```
<SELECTTO FROM="EmployeeInfo"
  WHERE="name"
  WHAT="*"
  LIST="MatchingEmployees"/>
```

In this case, `EmployeeInfo` is the name of the table that is being queried and is the name of the table that the resultset is cached against. Whenever the `EmployeeInfo` table is updated, CatalogManager flushes all the resultsets cached against it.

39.6.2 EXECSQL

`EXECSQL` lets you execute an inline SQL statement. You specify the table or tables that you want to cache the resultset against using the `TABLE` parameter. If you specify multiple tables (by using a comma-separated list), the resultset is cached against the first table in the list. Note that this means the resultset is cached based on the resultset cache settings specified for the first table, including timeout and maximum size.

CatalogManager deletes outdated resultsets as the specified tables are updated.

For example, the following query caches the resultset against the article table:

```
<EXECSQL SQL="SELECT article.headline, images.imagefile FROM article,images
WHERE article.id='FTX1EE17FWB' AND images.id='FTK9384FWW'" LIST="sqlresult"
TABLE="article,images"/>
```

39.6.3 CALLSQL

When you use the `ics.CallSQL` Java method, `CALLSQL` XML tag, or `ics:callsql` JSP tag to invoke a SQL query that is stored in the `SystemSQL` table, the table name is set by the query's entry (row) in the `SystemSQL` table.

The `SystemSQL` table has a `deftable` column that identifies the table name that the resultset from the query should be cached against. You can specify multiple tables by putting a comma-separated list of tables in the `deftable` column. The first table in the list is the table that the query is cached against.

Each query stored in the table must have a value in the `deftable` column. If it does not, CatalogManager cannot store the resultsets accurately, which means they cannot be flushed when it is necessary. Note that the table name must identify an existing table. If you enter the name of a table that does not exist yet or if you misspell the name of the table, the resultset cannot be cached correctly.

39.6.4 Search Forms in the WebCenter Sites Interface

The Search forms that you use to look for assets in the WebCenter Sites interface search by asset type. The resultsets from the search form queries are stored against the primary storage table for assets of that type.

For example, for the avisports sample site asset type named Article, those resultsets are cached against the `AVIArticle` table, for page assets it is the `Page` table, and so on.

39.6.5 Query Asset

Query assets can return assets of only one type. When you create a query asset, you specify what kind of asset the query asset returns in the **Result of Query** field: articles, imagefiles, and so on.

When that query asset is used on a page in the online site, WebCenter Sites stores the resultset against the table name of the primary storage table for the asset type that the query asset returns, for example, `Article` or `Imagefile`.

39.6.6 SEARCHSTATE

The `SEARCHSTATE` XML and JSP tags create a set of search constraints that are applied to a list or set of flex assets (created with the `ASSETSET` tags). A constraint can be either a filter (restriction) based on the value of an attribute or based on another searchstate (called a nested searchstate).

You can use the `SEARCHSTATE` and `ASSETSET` tags to extract and display flex assets or flex parent assets (not definitions or flex attributes) on your online pages for your visitors.

WebCenter Sites caches the resultsets of searchstates against the `_Mungo` table for the flex asset type. For example, if the searchstate returns the avisports sample site flex asset named article, the resultset is cached against the `AVIArticle_Mungo` table.

When you configure the delivery system, be sure to add resultset caching properties for all of your `_Mungo` tables.

39.7 Flushing the Resultset Cache

In most cases, data is written to the database through the CatalogManager API, which flushes the resultset cache when it is appropriate to do so. CatalogManager can flush all the resultsets cached against that table. Or, it can flush the resultsets cached against the tables that are written to. There are more possibilities for you to learn about.

For example:

- If you use WebCenter Sites Explorer to add a row to a table (the `SiteCatalog` table or the `ElementCatalog` table, for example), CatalogManager flushes all the resultsets cached against that table.

- If you use a form in the WebCenter Sites interface to add or edit an asset, a source, a category, a workflow process, a user, an ACL, etc., CatalogManager flushes the resultsets cached against the tables that are written to.

- If you use CatalogManager commands in an element of your own to update a single table, Catalog Manager automatically flushes the resultsets cached against that table.

- If you use CatalogManager commands in an element of your own to update multiple (joined) tables, Catalog Manager automatically flushes the resultsets cached against the joined tables.

- If you use the `CALLSQL` tag to execute a SQL statement that is stored in the `SystemSQL` table, Catalog Manager automatically updates the resultsets cached against the table or tables specified in the `deftable` column.

39.8 Switching Between Caching Frameworks

You can switch between the inCache and hash table frameworks by setting the `rsCacheOverInCache` property to either `true` or `false` in the `wcs_properties.json` file.

Resultset caching over inCache is enabled when the following conditions are met:

- The `linked-cache.xml` configuration file is placed in the application server's classpath (`WEB-INF/classes` directory).

- The `rsCacheOverInCache` property (in `wcs_properties.json`) is set to `true`.

39.9 About Resultset Caching Strategy and Properties

Before you configure resultset caching for your database, discuss with your team and database administrators how you should configure the database tables' properties to maximize the performance of the delivery system. The default properties control the table resultset caches. Table-specific properties enable you to fine-tune your systems for peak performance.

These topics describe the process of planning and using resultset caching properties for all tables and specific tables.

- Planning Your Resultset Caching Strategy

- Default Properties

- Table-Specific Properties

39.9.1 Planning Your Resultset Caching Strategy

Before you configure resultset caching for your database, create a spreadsheet of all the tables in your WebCenter Sites database, assemble a team of developers and database administrators, and discuss what the settings should be for all of your systems (development, management, testing, and delivery). One strategy is to identify default properties for a large group of similar tables, and then add table-specific properties for the exceptions. To fine-tune your delivery system for the best performance possible, however, it is likely that you will create custom properties for each table in the database.

> **Note:**
>
> If you set the `com.fatwire.logging.cs.cache.resultset` property, debugging messages about the resultset cache are written to the WebCenter Sites log file. Set the property in `logging-config.xml`.

39.9.2 Default Properties

This table describes resultset caching properties in WebCenter Sites `wcs_properties.json` that are assigned to all tables. The properties control the table resultset caches while no table-specific caching properties are assigned to the tables. The same properties are valid for resultset caching in both inCache and hash tables. Use the Property Management Tool, in the Admin interface, to change the values of the properties in the WebCenter Sites `wcs_properties.json` file. For information about using the Property Management Tool, see Introducing WebCenter Sites Tools and Utilities.

Table 39-1   Default Properties That Control the Resultset Cache

property	description
`cc.cacheResults`	Specifies the default number of resultsets to cache in memory. Note that this does not mean the number of records in a resultset, but the number of resultsets.
	Important: Unless you are debugging, do not set this property to `0` or `-1`. If you do, the WebCenter Sites interface will fail to save assets properly. (Setting this property to `0` or `-1` disables resultset caching for all tables that do not have their own caching properties configured.)
`cc.cacheResultsTime out`	Specifies the number of minutes to keep a resultset cached in memory.
	Setting this property to `-1` means there is no timeout value for tables that do not have their own caching properties configured.
`cc.cacheResultsAbs`	Specifies how expiration time in the resultset cache is calculated.
	• If this property is set to `true`, the expiration time is absolute. For example, if `cc.cacheResultsTimeout` is set to 5 minutes, then 5 minutes after the resultset was cached, it is flushed from the cache.
	• If this property is set to `false`, the expiration time is based on its idle time. For example, if `cc.cacheResultsTimeout` is set to 5 minutes, the resultset is flushed from the cache 5 minutes after the last time it was requested, rather than 5 minutes since it was originally cached.

39.9.3 Table-Specific Properties

Table-specific properties override the default properties and enable you to fine-tune your systems for peak performance.CatalogManager uses the default properties described in Table 39-1 and checks the `wcs_properties.json` file to determine if it contains any table-specific resultset caching properties.

You can create three resultset caching properties for each table in the WebCenter Sites database. Table-specific properties work in the same way as the default properties (described in Table 39-1).

Syntax for table-specific properties is as follows:

```
cc.<tablename>CSz=<number of resultsets>
cc.<tablename>Timeout=<number of minutes>
cc.<tablename>Abs=<true or false>
```

> **Note:**
>
> If an asset type is enabled for revision tracking, and you want to cache the resultsets of asset versions, use the properties above, but add `_t` after `<tablename>`:
>
> ```
> cc.<tablename>_tCSz=<number of resultsets>
> cc.<tablename>_tTimeout=<number of minutes>
> cc.<tablename>_tAbs=<true or false>
> ```
>
> See Resultset Caching Properties in the *Oracle Fusion Middleware Property Files Reference for Oracle WebCenter Sites*.

Open the Property Management Tool and add table-specific properties for each table that you want to control. See Property Management Tool.

40

Using Cache Management with WebCenter Sites

For all components to work together in your site, you must configure the rendering engine cache properly. Learn about some important caching concepts such as the rendering engine cache and its components. Also learn which cache configuration properties enable CacheManager to clear all caches (ContentServer, BlobServer, and Satellite Server caches) of any object that becomes obsolete because of changes in its underlying content.

Topics:

- About the WebCenter Sites Rendering Engine Cache

- About the CacheManager

- Enabling CacheManager

40.1 About the WebCenter Sites Rendering Engine Cache

The WebCenter Sites rendering engine cache is a two-tier cache. Tier 1 consists of ContentServer and BlobServer. And, tier 2 consists of Satellite Server.

Each component is independently configurable, with controls that fine-tune cache size, cache timeout, and dependency management behavior.

If the components are configured correctly, WebCenter Sites can effectively prevent users from viewing uncached content nearly all of the time. However, if these components are mis-configured, the behavior of WebCenter Sites can be non-intuitive and unpredictable. Inadequate caching can hamper performance, and improper co-ordination of the cache inventory can result in stale content being rendered. To address this, WebCenter Sites includes a module called CacheManager, which can actively manage the cache on behalf of the whole system.

40.2 About the CacheManager

CacheManager can record the existence of a *compositional dependency* against an object that is to be cached by the rendering engine. For example, if a pagelet renders an asset, then the asset is a compositional dependency on that page. If the asset changes, the page is no longer valid and must be flushed from cache.

Using CacheManager to flush the cache requires that you surrender full control over the lifecycle of rendering engine cache objects to CacheManager. You do this by specifying that the objects never expire from the cache. When CacheManager determines that they are obsolete because of changes in the underlying content (that is, in one of the compositional dependencies recorded against each object), it removes those objects from the cache.

> **Note:**
>
> When you specify an infinite expiration time, CacheManager keeps a record of all objects that are cached, and what dependencies are tracked against them. This record is stored on WebCenter Sites, and it is linked to the cached object on the first tier. This record enables CacheManager to infer the existence of objects in the second tier cache and therefore flush the objects from the second tier cache.
>
> If an object expires from the cache, its record is removed, leaving CacheManager without the information it requires to properly flush the object from the second tier cache.

CacheManager features are almost completely automatically enabled.

- By default, the cache is configured so that objects never expire.

- Compositional dependencies are recorded against the Blob and Page cache on the lower tier. Tags such as `<asset:load>` and `<render:sateliteblob>` provide *automatic* compositional dependency recording (see the *Oracle Fusion Middleware Tag Reference for Oracle WebCenter Sites Reference* for a complete list), whereas the two tags `<portal:logdep>`, and `<render:logdep>` provide *explicit* compositional dependency recording.

- Whenever assets are modified or published, WebCenter Sites automatically invokes CacheManager to purge the old content from the cache and, in the case of publishing, instructs CacheManager to pre-cache the new content in the background before flushing the second tier cache.

Site visitors enjoy the best possible cache performance when they only view cached content.

40.3 Enabling CacheManager

Learn about Tier 1 properties that regulate Page and BlobServer blob cache and Tier 2 properties that deal with the Satellite Server page and blob cache.

These topics describe the Tier 1 and Tier 2 cache configuration properties and how they must be set to enable CacheManager.

- Tier 1 Cache Configuration Properties

- Tier 2 Cache Configuration Properties

40.3.1 Tier 1 Cache Configuration Properties

The tables in this section describe properties that regulate the page cache and BlobServer blob cache.

See the *Oracle Fusion Middleware Property Files Reference for Oracle WebCenter Sites*.

The following table describes page cache properties:

Table 40-1   WebCenter Sites Page Cache Properties

Property	Description
cs.IItemList	This property specifies the class implementing the IItemList interface. An illegal value results in CacheManager having no effect.

The following table describes BlobServer cache properties:

Table 40-2   BlobServer Cache Properties

Property	Description
bs.bCacheTimeout	This property specifies how many seconds a blob should remain cached by BlobServer.
bs.bCacheSize	This property specifies how many blobs will be stored in the BlobServer cache. This has no effect on CacheManager.
cs.recordBlobInventory	This property specifies whether compositional dependencies should be recorded against blobs. This property must be set to true (the default) for CacheManager to operate on blobs.
bs.security	This property controls the security feature of BlobServer. When BlobServer security is enabled, caching is disabled. Consequently, BlobServer security is incompatible with CacheManager's Intelligent Cache Management features. By default, this level of security is disabled.

40.3.2 Tier 2 Cache Configuration Properties

Tier 2 cache configuration properties deal with the Satellite Server cache, both page and blob.

None of the Tier 2 properties affect the correct operation of CacheManager. They do, however, serve as important diagnostic aids if CacheManager is operating incorrectly. The timeout and configuration values of the Tier 2 cache properties are important in troubleshooting unpredictable behavior.

Typically, unpredictable behavior results when objects are cached on the Tier 2 cache but not on the Tier 1 cache, and so they are not actively flushed when the dependent asset is saved or published. See the *Oracle Fusion Middleware Property Files Reference for Oracle WebCenter Sites* for configuration details.

Unpredictable behavior can also result if no compositional dependency is recorded against an object that is cached. This scenario precludes all active management of that object in the caches. See the *Oracle Fusion Middleware Tag Reference for Oracle WebCenter Sites Reference* for details about which tags automatically record compositional dependencies, and which tags must be used in conjunction with explicit recording using one of the :logdep tags.

> **Note:**
>
> Do not record excessive compositional dependencies on your pages or blobs. This causes unnecessary flushing of the cache, which under certain circumstances can result in severe performance problems during publishing. Be very careful when recording unknown compositional dependencies. See Coding Elements for Templates and CSElements.

41

Using Advanced Page Caching Techniques

Advanced page caching techniques include caching pages and blobs, where they are cached, and how they are retrieved from cache on both WebCenter Sites and Satellite Server systems.

Topics:

- About Advanced Page Caching

- Configuring the WebCenter Sites Cache

- Configuring the Blob Server Cache

- Configuring the Satellite Server Cache

- CacheInfo String Syntax

- Caching Best Practices

41.1 About Advanced Page Caching

Caching improves the speed at which WebCenter Sites serves pages. The caching-rendered content doesn't need to be re-rendered at each request. This improves the response time.

The caching system has multiple layers. This allows cached objects to be regenerated on one cache level, while the client is being served cached content from another cache level. WebCenter Sites is the inner level of cache, and Satellite Server is the outer layer of cache.

41.2 Configuring the WebCenter Sites Cache

Both WebCenter Sites and Satellite Server cache pages, pagelets, and blobs. WebCenter Sites provides three different rendering engine caches: CS page cache, BlobServer cache, and SS cache.

All the caches can be configured and emptied as follows:

- For information on setting expiration times for WebCenter Sites cache and BlobServer cache, see Setting Expiration Time for an Individual Entry.

- For information on removing objects from the cache for WebCenter Sites cache, BlobServer cache, and Satellite Server cache, see Explicitly Removing Entries from Cache.

There are two levels of caching for the WebCenter Sites page cache:

- In the database.

- In memory. Memory cache is a transparent subset of the database cache; however, it is independently configurable.

See these topics:

- Setting Expiration Time for an Individual Entry
- Explicitly Removing Entries from Cache

41.2.1 Setting Expiration Time for an Individual Entry

The lifetime of an entry in the page cache is determined by the `cscacheinfo` setting. The `CacheInfo` object derives values that are not explicitly set in the **cscacheinfo** field from the configuration file. For `CacheInfo` syntax, see CacheInfo String Syntax.

41.2.2 Explicitly Removing Entries from Cache

WebCenter Sites provides two ways of removing entries from cache: manually and automatically, using CacheManager.

41.2.2.1 Manual Removal

You can remove entries from the page cache manually using the CacheServer servlet. The CacheServer provides two options:

- Flush the entire cache.

- Force a flush of all pages at the moment they expire. To invoke the CacheServer *flush all* functionality, you must be logged in as a user with *destroy* privileges on the `SiteCatalog` table, and specify the parameter `all=true` when invoking the CacheServer servlet. If you do not specify a parameter, then all expired entries (those whose expiry date is in the past) are cleared from the cache immediately. Entries that have not yet expired are not cleared.

Note:

In no case will an expired entry be served from the cache, even if it is still in the database table. WebCenter Sites checks the expiry date of any page it retrieves from cache before serving the page. If WebCenter Sites attempts to serve a page that has expired, the page will be removed from the cache immediately and a new page will be generated.

The CacheServer URL is in the following format: `http://{hostname}:{port}/{context}/CacheServer?all=true&authusername=fwadmin&authpassword=xceladmin`

- `Hostname`: The name of server that hosts the CacheServer servlet.

- `Portnumber`: The port number of the server that hosts the CacheServer servlet.

- `authusername`: The username of the user who is authorized to access the servlet.

- `authpassword`: The password of the user who is authorized to access the servlet.

41.2.2.2 Automatic Removal

CacheManager is a module closely tied into the WebCenter Sites page and blob cache mechanisms. It lets you manage the contents of all of the rendering caches based on the items loaded on a page, on the expiration of the pages, or on parameters passed into pages.

CacheManager alone could be the subject of an independent document. An overview of its functionality is provided here. For complete information about methods and required arguments, consult the `COM.FutureTense.Cache.CacheManager` Javadoc.

1. `CacheManager` is instantiated using one of two constructors. One constructor sets `CacheManager` with all of the currently registered Satellite Servers. The other constructor lets you specify which Satellite Servers this instance of `CacheManager` is to manage.

2. Next, `CacheManager` needs to be populated with pages and blobs. This is done by using one of the following methods:

```
setByCachedDate(ICS ics, boolean before, String timestamp)
setByItemDate(ICS ics, boolean before, String timestamp)
setPagesByArg(ICS ics, String paramName, String paramValue)
setPagesByID(ICS ics, String[] ids)
```

The contents of the Page, Blob and Satellite caches are closely tied together. It is always the case, except as a result of a configuration error, that any object cached on Satellite Server is present in the WebCenter Sites cache. This means that WebCenter Sites has a record of all entries in all rendering engine caches. **CacheManager** uses this record to manage the contents of each of the caches, without having to directly interrogate each cache for the information explicitly.

```
setByCachedDate(ICS ics, boolean before, String timestamp)
```

This method lets you populate CacheManager based on the date an entry was last added to the cache. You can choose whether you want to populate it with all of the entries modified either before or after the date specified.

```
setByItemDate(ICS ics, boolean before, String timestamp)
```

This method lets you populate CacheManager based on the date an item on an entry was last modified. As with `setByCachedDate(ICS, boolean, String)`, you can choose whether you want all entries whose items were modified before or after the date specified.

```
setPagesByArg(ICS ics, String paramName, String paramValue)
```

This method lets you populate CacheManager based on name-value pairs present in the cache key (including pagename).

```
setPagesByID(ICS ics, String[] ids)
```

This method lets you populate CacheManager based on the exact item IDs of the items stored on the pages or blobs in the cache.

Once fully populated, CacheManager is able to manage the contents of the caches. This is done using one of the four main service methods:

- `flushCSEngine(ICS ics, int mode)`

This method flushes all of the pages and blobs currently populated in the CacheManager from the WebCenter Sites page and blob caches.

- `flushSSEngines(ICS ics)`

  This method flushes all of the pages and blobs currently populated in the CacheManager from the Satellite Server cache. This is done by sending an HTTP request to the FlushServer servlet with the appropriate `<page>` and `<blob>` tags embedded in it. Satellite Server interprets these tags and converts them into a cache key, then flushes the corresponding pages from cache.

- `refreshCSEngine(ICS ics, int mode)`

  This method sends a request (using `ICS.ReadPage` or `ICS.BlobServer`) that regenerates the object and automatically re-populates the cache.

- `refreshSSEngines(ICS ics)`

  This method sends a request using HTTP to Satellite Server to read the pages. The returned bytes are ignored, but the result is that the Satellite Server cache is re-populated.

You can use these methods to take advantage of double-buffered caching, a tool that can enable extremely high performance dynamic sites. See Understanding Page Design and Caching.

41.3 Configuring the Blob Server Cache

The BlobServer cache is an *all or nothing* cache. Its entries are either globally cached or globally not cached. BlobServer caching is disabled if security is enabled, or if `bs.security=true`.

See these topics:

- Consideration About Configuring Maximum Cache Size

- Setting Expiration Time for an Individual Entry

- Explicitly Removing Entries from Cache

41.3.1 Consideration About Configuring Maximum Cache Size

The property `bs.bCacheSize` in the WebCenter Sites `wcs_properties.json` file specifies the number of entries the blob cache contains. If the size is set to a negative number, the blob cache is allowed to grow indefinitely.

41.3.2 Setting Expiration Time for an Individual Entry

Blob Server does not support expiration for cached entries. All cached objects reside in cache for the timeout determined by the `bs.bCacheTimeout` property in the WebCenter Sites `wcs_properties.json` file. A negative timeout indicates that entries should not time out. A positive integer specifies the number of minutes an object resides in cache.

41.3.3 Explicitly Removing Entries from Cache

You can use BlobServer to flush either individual entries or all entries from the cache.

41.3.3.1 Manual Removal

To remove an entry from cache manually, rename the `blobtable` parameter to `flushblobtable`. This removes the entry corresponding to the rest of the parameters from the cache.

To remove all entries from the cache manually, there are two options.

- Invoke the BlobServer servlet with the parameter `flushblobtables` (notice the "s").

- Invoke the CacheServer servlet as described above. Note that this flushes all pages and all blobs from the cache.

41.3.3.2 Automatic Removal

Because blob dependency items are recorded when blob links are generated, you may invoke CacheManager to manage blobs and pages. CacheManager always manages blobs and pages together. See CacheManager, About the CacheManager, and Enabling CacheManager.

> **Note:**
>
> Developers should check the asset status before serving blobs. This avoids issues such as the BlobServer serving a blob after an asset has been voided (this behavior occurs with basic assets only).

41.4 Configuring the Satellite Server Cache

You configure the generic Satellite Server cache in the `wcs_properties.json` file.

However, cache configuration is often overridden on an object-by-object basis.

41.4.1 Configuring Maximum Cache Size

- Update the `cache_max` property in the `wcs_properties.json` file to the maximum number of entries that can be stored in the cache at once.

  If you set the property to a negative integer, the cache is not limited by size. Any positive integer specifies the maximum number of entries that can be stored in the cache.

41.4.2 Explicitly Removing Entries from Cache

You can remove individual entries from the Satellite Server cache either manually or using CacheManager, as explained in this section.

- **Manual Removal**: Satellite Server includes a servlet called FlushServer. By submitting a `GET` request to this servlet, specifying the user name, password and reset parameters, it is possible to flush all of the contents of the Satellite Server cache. It is not possible to flush individual entries using `GET`.

- **Automatic Removal**: It is possible to flush the Satellite Server cache using CacheManager. CacheManager is only able to flush entries on Satellite Server if a corresponding object is cached on WebCenter Sites. This is because of the way WebCenter Sites tracks the contents of the Satellite Server cache.

When the Satellite Server cache is flushed by using CacheManager, a corresponding object is cached on WebCenter Sites. The corresponding object is required because of the way WebCenter Sites tracks the contents of the Satellite Server cache.

The relevant CacheManager methods for dealing with the Satellite Server cache are `flushSSEngines()` and `refreshSSEngines()`. See Explicitly Removing Entries from Cache.

41.5 CacheInfo String Syntax

The **cscacheinfo** and **sscacheinfo** fields of the SiteCatalog are populated with a CacheInfo string. Learn how the format of the two-part, comma-separated string determines whether the page is to be cached and when it expires.

Sample values are as follows:

```
false
true
true,*
true,~4
true,@1987-06-05 04:32:10
true,#00:00:00 */*/*
*
(blank)
```

CacheInfo String: First Part

The first part in CacheInfo must be one of the following values:

```
false
true
(blank)*
```

- If the value is `false`, then the page is not cached.

- If the value is `true`, then the page is cached according to the information provided in the second element.

- If the value is blank, then the `wcs_properties.json` property `cs.alwaysusedisk` is checked. If this property is set to `yes`, then a blank value is interpreted as having the same behavior as `true`. If the value is set to `no` (the default value), then a blank value is interpreted as having the same behavior as `false`.

- If the value is *, then it is treated as blank.

CacheInfo String: Second Part

This describes when a page that is to be cached should be removed from cache. If the first element is `false` (or is interpreted as `false`), then the second element is ignored.

There are three ways of specifying the expiration of a page:

```
page timeout (in minutes)
instant in time expiration
cron-like TimePattern expiration
```

Legal values include:

```
~<number of minutes>
@<date in JDBC format>
#<COM.FutureTense.Util.TimePattern format>
```

```
*
(blank)
```

Page Timeout

If the second element starts with the tilde symbol (~), then the following value must be an integer. The value of this integer is the number of minutes a page will remain in cache after it was first created. A negative value or `0` indicates that the page will never expire (it will remain in cache forever).

Absolute Moment in Time

If the second element starts with the at symbol (@), then the following value must be a date expressed in the JDBC date string format, that is, `YYYY-MM-DD HH:MM:SS`. Once that date has passed, cached pages are flushed from cache and the page is no longer cached.

TimePattern

The TimePattern format is supported for describing page cache expiration. If the second element starts with a hashtag (#), then the following value must be a valid TimePattern string as defined by the public class `COM.FutureTense.Util.TimePattern`.

In general, the TimePattern syntax corresponds to the format used in most UNIX cron tables. It lets you specify expiration at a specific time or times every day, month, week, day of week, and year.

The TimePattern format is expected to become the most widely used format for page expiration.

Wildcard

If the second element is *, then the page will assume a timeout expiration behavior, as described in Page Timeout. The timeout value is read from the `wcs_properties.json` file `cs.pgCacheTimeout` property.

Blank

If the second element is blank, then it assumes the same behavior as *.

41.6 Caching Best Practices

Ideally, a WebCenter Sites webpage should have a maximum of six to ten cached primary pagelets served by Remote Satellite Server. Too many cached pagelets lead to overcaching and adversely affect the system performance.

These topics describe practices that prevent overcaching and improve response times and throughput:

- Few Pagelets Per Page

- Share Cache Between Pages

41.6.1 Few Pagelets Per Page

- Design page templates in such a way that there are fewer uncached pagelets per page. The default `style=pagelet` from the `<render:calltemplate>` tag is not always the best choice. Fewer pagelets per page improve response times and throughput.

- Switch from Remote Satellite Server to Co-Resident Satellite Server if you already have page templates and redesigning them is not feasible. No roundtrips are required between the WebCenter Sites servlet and Co-Resident Satellite Server. Co-Resident SatelliteServer is likely to give better response times than Remote Satellite Server when there are more uncached pagelets on a page.

- Reduce the number of cached pagelets by combining them. First check if each `<render:calltemplate>` call is using the `pagelet` default style attribute value. Reduce the number of pagelets by inlining the `<render:calltemplate>` calls as element calls (`style=element`).

  The typical use case for specifying `style="element"` is when the outer calling template (whether page or pagelet) shares the same `c`, `cid` as an inner called pagelet template. Since both cached objects expire at the same time (that is, they have the same asset dependencies), there is no advantage in caching them separately.

- Rather than caching subpagelets directly on Remote Satellite Server, back up subpagelets through pagelet cache on the ContentServer servlet by using the `style="embedded"` variant. That is, inline the `<render:calltemplate>` calls as ContentServer page fragments (`style=embedded`).

  For example, if a list of product summaries in a webpage contains ten items, it means the default `style="pagelet"` is applied to each of them. Due to the pagelet style, ten roundtrips between the WebCenter Sites servlet and Remote Satellite Server would be required to render this list. However, if existing `<render:calltemplate>` calls for the subpagelets are set to use `style="embedded"` -- while ensuring to cache the outer list template itself -- then just one roundtrip for the entire product list pagelet is required. This is because only the outer template is served by Remote Satellite Server with the remaining subpagelets only served back on the ContentServer servlet.

41.6.2 Share Cache Between Pages

When the `cid` is passed to every cached pagelet, the cache is not shared between webpages. For example, a site may contain 5,000 webpages, all of which pass the `cid` to the leftnav. There would also be 5,000 leftnavs cached independently of the calling webpage. This scenario can produce two situations:

- When a single Page asset in the site navigation tree is edited and published, all 5000 leftnavs uncache. This is because the Page asset just published has logged a dependency to all leftnav pagelets.

- When a single Article asset is edited and published, both the cached outer webpage (typically the Layout template) and the leftnav are uncached and need to be re-evaluated. Because cid is passed to leftnav, Content Server uncaches outer webpage and leftnav as well. This approach is not efficient since the Article has nothing to do with the Site Navigation hierarchy.

To avoid overcaching in these situations, change the leftnav so that it always takes a Page `cid`, and not just any `cid`. The calling template should be able to determine what Page it belongs to so that it can pass that as a cache argument. Only the Page `cid` should log a cache dependency to the pagelet and no other Page assets. Pseudocode is as follows:

```
If c is not "Page"
then
    Calculate the site navigation PageNode by doing a reverse look up of the Page
```

```
from the current asset's cid.
    Alternatively, have the site navigation Page as an attribute of the current
asset and simply fetch it.
        then calculate the PageNode from that
else
    Calculate the PageNode from the current Page cid.
```

This way, the cached version of this pagelet is bound to the owner's related Page asset alone. When a single Page asset is edited in the site navigation tree and published, only a single leftnav, whose `cid=<the current Page asset>`, will uncache. When a single Article asset is edited and published, only the cached outer webpage (typically the Layout template) needs to be re-evaluated.

When editors are updating and changing the system on a daily basis, developers' choices have a huge impact on the efficiency of the system. Without the suggested improvement, even editing a single asset can cause the system to do a lot more work at publish time than required. The arguments that are explicitly passed to each pagelet have a direct impact on overall performance.

Part VII

Migrating Your Work to Your Content Management System

Get introduced to the process of importing assets with the XMLPost utility, and also to the posting elements that you use with the XMLPost utility and the BulkLoader utility.

- Importing Assets of Any Type

- Importing Flex Assets

- Importing Flex Assets with the BulkLoader Utility

42

Importing Assets of Any Type

You can import assets of all types into the WebCenter Sites database using the
XMLPost utility. This utility is based on the WebCenter Sites `FormPoster` Java class,
and it is delivered with the WebCenter Sites base product. It imports data using
the `HTTP POST` protocol.

Topics:

- About Importing Assets Using the XMLPost Utility

- Using XMLPost Configuration Files

- Using XMLPost Source Files

- Using the XMLPost Utility

- Customizing RemoteContentPost and PreUpdate

- Troubleshooting XMLPost

For information about importing your assets when you are using the flex asset data
model, see Importing Flex Assets.

42.1 About Importing Assets Using the XMLPost Utility

After you have determined your data design, created your asset types, tested them on
your development system, and moved them to your management system, you're
ready to import assets (content) from their current source into the database on the
management system. You may also have remotely generated content (generated using
a wire feed service or some other source) that you would like to import into the
WebCenter Sites database on your management system.

You use the XMLPost utility to import any data into the WebCenter Sites database. To
import assets, you instruct the XMLPost utility to invoke one of the importing
(posting) elements provided by WebCenter Sites, as appropriate for that asset type.

There are four components involved in this process:

- The XMLPost utility, which is delivered with WebCenter Sites.

- A posting element. WebCenter Sites delivers a posting element named
  `RemoteContentPost`. WebCenter Sites delivers three additional posting
  elements, described in Importing Flex Assets.

- A configuration file with an `.ini` file extension. You create a configuration file for
  each asset type that you plan to import. This file contains information about what
  to expect in the source files (what tags XMLPost will find there), what to do with
  the data provided, and which importing (posting) element to use to import the
  data.

- • The source files. You provide an individual source file for each asset that you want to import (well-formed XML files). Each tag in a file identifies a field for that asset type. The information contained in the tag is the data to be written to that column.

The XMLPost utility parses the configuration file to determine how to interpret the data provided for the asset type. It parses the source files and creates name/value pairs for each field value, and passes those name/value pairs as ICS variables to the `RemoteContentPost` element. The `RemoteContentPost` element then creates the asset from the variables.

You can also create your own posting elements that work with the XMLPost utility. However, for importing assets, the posting elements that are provided by WebCenter Sites should meet your needs.

> **Note:**
>
> For added security, you must rename the `RemoteContentPost` page to prevent attempts to hack into the system.

42.1.1 What the Developer Does

This section provides a brief overview of the steps that the developer completes before invoking the XMLPost utility and what the XMLPost utility does.

General steps that you perform when you import assets into your WebCenter Sites database:

1. Create a configuration file that identifies the type of asset that is to be imported and the tags that are used in the source files.

   This file also sets several configuration properties, including the name of the SiteCatalog entry for the posting element that you want XMLPost to use. For all assets, the name of this posting element is `RemoteContentPost`. For information about the posting elements for flex assets, see Importing Flex Assets.

   Note that the configuration file is specific for this asset type. You must provide a separate configuration file for each asset type.

2. Create the source files for the data that you want to import. Create a separate source file for each individual asset.

3. Place the source and configuration files in a directory on the management system.

4. From that directory, invoke the XMLPost utility, identifying the source files and the configuration file to use for those source files.

42.1.2 What XMLPost and WebCenter Sites Do

After you invoke the XMLPost utility to import the source files, the process begins as shown in the following figure and in the list of steps that follows.

Figure 42-1   XMLPost Utility Process Diagram
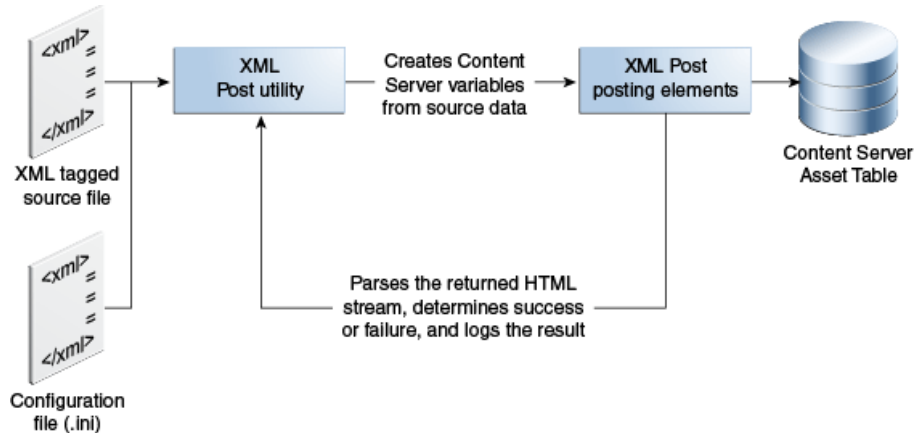
1. The XMLPost utility parses the configuration file.

2. XMLPost parses the source file and creates name/value pairs for each field value specified in the source file.

3. XMLPost invokes the `FormPoster` Java class by posting (`HTTP POST`) the name/value pairs as `ICS` variables to the pagename passed in from the configuration file. When you are importing basic asset types, that pagename is:

    `OpenMarket/Xcelerate/Actions/RemoteContentPost`

4. WebCenter Sites locates the page in the `SiteCatalog` table and invokes the root element of the `RemoteContentPost` page, which has the same name by default (`RemoteContentPost`).

5. The `RemoteContentPost` element passes the data from the source files as variables to the `PreUpdate` element for assets of that type.

6. The `PreUpdate` element sets the variable values for that asset and then returns to the `RemoteContentPost` element.

7. The `RemoteContentPost` element creates the asset.

8. The web server returns a stream of HTML to XMLPost, which then parses the stream to determine whether the import operation succeeded or failed, logging the results to a text file that you specify in the configuration file.

9. If the asset type of the asset that you are importing uses a search engine, `RemoteContentPost` indexes the new element.

10. If you set a certain parameter in the configuration file, `RemoteContentPost` deletes the source files for the assets that were successfully imported.

42.2 Using XMLPost Configuration Files

An XMLPost configuration file includes three types of properties. Some properties provide database and environment information to XMLPost, some import configuration values, and some specify the tags used in the source files.

-   Properties that provide information to XMLPost about the database and environment remain the same even if you create your own posting element.

- Properties that provide configuration values for the posting (importing) process. The following topics describe properties that you must provide for the `RemoteContentPost` element to function correctly.

  Examples of properties include the URL of the page that invokes `RemoteContentPost`, a user name and password that gives XMLPost write privileges to the asset type table in the database, the name of the asset type that you want to import, how to log errors, and any data values that are the same for all of the assets that you are importing.

- Properties that specify the tags that are used in the source files.

Certain information, such as which site the assets should belong to or which workflow should be assigned to the asset, can be configured either in the `RemoteContentPost` section of the configuration file or the source file section.

For example, when working with a single content management site or importing assets that belong to the same site, specify the name of the site in the configuration section to avoid repeating that information in each source file. For multiple content management sites, specify which sites an asset belongs to in the individual source files.

This section includes the following topics:

- Configuration Properties for XMLPost

- Configuration Properties for the Posting Element

- Configuration Properties for the Source Files

- Sample XMLPost Configuration File

42.2.1 Configuration Properties for XMLPost

This table lists the properties that specify database connection information and other general configuration instructions that the XMLPost utility needs.

Table 42-1   Configuration Properties for XMLPost

Property	Description
`xmlpost.xmlfilename filter`	Required. The file extension for your source files. Typically set to `xml`. For example: `xmlpost.xmlfilenamefilter: .xml`
`xmlpost.proxyhost`	Optional. If a firewall separates you and the WebCenter Sites database that you want to import the assets in to, use this property to specify the host name of the proxy server. For example: `xmlpost.proxyhost: nameOfServer`

Table 42-1   (Cont.) Configuration Properties for XMLPost

Property	Description
xmlpost.proxyport	Optional. If a firewall separates you and the WebCenter Sites database that you want to import the assets in to, use this property to specify the port number on the proxy server that XMLPost should connect to. For example: xmlpost.proxyport: 80
xmlpost.url	Required. The first part of the URL for the page entry of the posting element. XMLPost creates the URL for the posting element by prepending the value specified for this property to the value specified for the pagename postargname (described below). The value that you set for this property should use the following convention: • The name of the server that holds the WebCenter Sites database. • The CGI path appropriate for the application server software installed on the server. For WebLogic and WebSphere this path is /servlet/. • The name of the ContentServer servlet. For example: xmlpost.url: http://servername/servlet/ ContentServer
xmlpost.logfile	Optional. The name of the file to log the results of importing (posting) each source file. Each source file is posted to the WebCenter Sites database through a post request. When the post request returns from the web server, XMLPost parses the HTML stream that the web server returned, searching for the postsuccess and postfailure parameters. XMLPost then writes the result to the file that you name identify with this parameter. For example: xmlpost.logfile: ArticlePost.txt
xmlpost.success	Optional. The string to look for in the response to determine if the post was a success. For example: xmlpost.success: Success!
xmlpost.failure	Optional. The string to look for in the response to determine if the post was a failure. For example: xmlpost.failure: Error

Table 42-1   (Cont.) Configuration Properties for XMLPost

Property	Description
xmlpost.deletefile	Optional.
	Whether to delete the source files after they have been successfully imported into the WebCenter Sites database. Valid settings are y (yes) or n (no). By default, the source files are not deleted.
	For example:
	xmlpost.deletefile: y

42.2.2 Configuration Properties for the Posting Element

This table lists the arguments that specify information that must be posted to the RemoteContentPost page (and passed to the RemoteContentPost element). The values of these arguments are concatenated into the URL that is posted to the RemoteContentPost page. These arguments can be in any order in the configuration file.

Table 42-2   Configuration Properties for the Posting Element

Property	Description
xmlpost.numargs	Required.
	The page name is the primary variables out of several required variables that the configuration file passes to XMLPost as name/value pairs attached to the URL. Use this property (xmlpost.numargs) to tell XMLPost how many variables the configuration file is passing in.
	For example:
	xmlpost.numargs: 7
	Note that you can also specify your own custom variables with these name/value pairs.
xmlpost.argname1: pagename	Required.
	The pagename for the RemoteContentPost element. Typically the pagename argument is specified as xmlpost.argname1.
	For example:
	xmlpost.argname1: pagenamexmlpost.argvalue1: OpenMarket/Xcelerate/Actions/RemoteContentPost

Table 42-2   (Cont.) Configuration Properties for the Posting Element

Property	Description

`xmlpost.argname2:` `AssetType`	Required.
	
	The asset type of the assets that are defined in the source files. Typically, `AssetType` is specified as `xmlpost.argname2`.
	
	For example:
	
	`xmlpost.argname2: AssetType`
	
	`xmlpost.argvalue2: Collection`
	
	Note that the value for the `AssetType` argument must exactly match the table name of the table that holds assets of this type.

`xmlpost.argname3:` `authusername`	Required.
	
	The user name that you want XMLPost to use to log in to the WebCenter Sites database into which you are importing the assets. Typically, `authusername` is specified as `xmlpost.argname3`.
	
	For example:
	
	`xmlpost.argname3: authusername`
	
	`xmlpost.argvalue3: editor`
	
	The user name that you specify must have permission to write to the table that holds assets of the type that you are importing. (That is, it must have the appropriate ACLs assigned to it.)

`xmlpost.argname4:` `authpassword`	Required.
	
	The password for the user that XMLPost logs in as to the WebCenter Sites database into which you are importing the assets. Typically, `authpassword` is specified as `xmlpost.argname4`.
	
	For example:
	
	`xmlpost.argname4: authpassword`
	
	`xmlpost.argvalue4: xceleditor`

`xmlpost.argname5:` `xmlpostdebug`	Optional
	
	Whether to include debugging information with the results information that is written to the XMLPost log file identified with the `xmlpost.logfile` property.
	
	You can set this property to any value. For example:
	
	`xmlpost.argname5: xmlpostdebug`
	
	`xmlpost.argvalue5: on`
	
	Note: Be sure to include a value for the `xmlpost.logfile` property if you enable debugging.

Table 42-2   (Cont.) Configuration Properties for the Posting Element

Property	Description

`xmlpost.argname6: inifile`	Optional.
	
	The name of the `ini` file to use when connecting to the WebCenter Sites database. Typically, `inifile` is specified as `xmlpost.argname5`.
	
	For example:
	
	`xmlpost.argname6: inifile`
	
	`xmlpost.argvalue6: futuretense.ini`

`xmlpost.argname7: publication`	Optional.
	
	Although using this property is optional, you must specify a site for each asset that you are importing.
	
	If your system uses one content management site (publication), or if all assets of this type should be enabled on the same site, use this argument to set the name of the site.
	
	For example:
	
	`xmlpost.argname7:`
	`publicationxmlpost.argvalue7: Fiscal News`
	
	You must specify the value for site for each asset in the individual source files when using multiple content management sites.

`xmlpost.argname8:startmenu`	Optional.
	
	If you are using workflow and you want the same workflow assigned to all of the assets that you are importing, use this argument to set the **Start Menu** shortcut for the assets. (It is a **Start Menu** shortcut that assigns a workflow ID to a new asset.)
	
	For example:
	
	`xmlpost.argname8: startmenu`
	
	`xmlpost.argvalue8: New Article`
	
	If you have multiple workflows for assets of this type, you must specify the value for the **Start Menu** shortcut for each asset in the individual source files.

42.2.3 Configuration Properties for the Source Files

The source file section in a configuration file specifies which tags are used in the source files. A tag represents a column name in the table that holds assets of this type. The content between a pair of tags is the information that is to be written to that column. Configuration files must list a tag for each column in the asset type's primary storage table, which is why you must provide a separate configuration file for each asset type.

This section includes the following topics:

- Site Properties

- Asset Type Properties

42.2.3.1 Site Properties

In addition to the tags available for your asset types, The following table describes more tags that let you specify the sites an asset should be associated with and the workflow it should use.

Table 42-3   Site Properties

Site tag property	Value	Description
postpublication	y or n (yes or no)	Optional. Specifies that a source file will provide a site name that identifies which site the asset belongs to. For example: `postpublication: y` Note that a site (publication) value provided in a source file with the `publication` tag overrides the value specified for a publication argument in the `XMLPost` section of the configuration file.
postprimarypubid	y or n (yes or no)	Optional. Specifies that a source file will provide a value for `pubid` (a unique ID for the site) that identifies which site the asset belongs to. For example: `postprimarypubid: y`
postpublist	y or n (yes or no)	Optional. Specifies that a source file will provide a list of sites that the asset is shared with. For example: `postpublist: y`
poststartmenu	y or n (yes or no)	Optional. Specifies that a source file will provide a value for the **Start Menu** short cut that places the asset into a workflow process. For example: `poststartmenu: y`

When the site or the workflow is the same for all of the assets that you are importing, specify the value for site or workflow as an argument in the `XMLPost` section of the configuration file. That way, you do not have to duplicate the same information in all of the source files.

42.2.3.2 Asset Type Properties

To set up the tags that are specific to your asset types, you specify a tag for each column in the database table for assets of that type. However, the source files are not required to include data tagged with every tag in the configuration file. (Of course, they must include data for required fields.)

For each tag representing a field (column), you specify the name of the tag and optionally some additional processing properties for the tag. The name of the tag is the

name of the field (column). For the additional properties, the convention is a word prepended to the name of the tag.

This table describes how to specify the tags that are specific to your asset types:

Table 42-4   Asset Type Properties

Tag property	Value	Description
post*tagname*	y or n (yes or no)	Required. Specifies the name of the tag. The name should exactly match the name of the field that it represents. For example, the tag property for a name field is: `postname: y`
trunc*tagname*	*N* (integer)	Optional. Whether to truncate the data in the source file marked by this tag. For example: `truncname: 64` If XMLPost finds a string in the `<name>` tag that exceeds 64 characters, it shortens it to 64 characters and stores the truncated string in the variable.
notrim*tagname*	y or n (yes or no)	Optional. Whether to trim the white space at the beginning or end of the tag. To keep the white space, set this property to y (yes). For example: `notrimname: y` Leave it blank to let XMLPost trim the white space for the tag by default.

Table 42-4   (Cont.) Asset Type Properties

Tag property	Value	Description
multi*tagname*	combine or separate	Required if the same tag is used more than once in a single source file. Determines how many variables to use for the data when a tag is used more than once in the source file. Set it to combine, to store the data from all of the tags in the same variable with commas separating each value (a comma delimited string). Set it to separate, to store the data from each tag in a separate variable. Those variables are identified by appending the value that you set for seed*tagname* to the variable name. For example, for a keyword field (column): • If you set multikeyword: combine, XMLPost stores all the values marked by a keyword tag to the same keyword variable. • If you set multikeyword: separate and seedkeyword: 1, XMLPost stores each value in a separate variable. The first value it finds is stored in a variable named keyword1. The second value is stored in a variable named keyword2, and so on.
seed*tagname*	*seed value*	Required when multi*tagname* is set to separate. The number to start at when XMLPost increments the suffix assigned to variable names, when a tag is used more than once and you do not want the data contained in those tags written to the same variable. See the description of multi*tagname*. For example: multikeyword: separate seedkeyword: 1

Table 42-4 (Cont.) Asset Type Properties

Tag property	Value	Description
file*tagname*	y or n (yes or no)	Required if the tag represents an upload field (a URL column or BLOB).
		If the tag represents a field that has a URL column, you must include this property and the source file must specify the name of the file that RemoteContentPost is to upload to that column.
		For example, an imagefile asset type has an upload field named urlpicture. A configuration file for the imagefile asset type will contain the following properties:
		posturlpicture: y
		fileurlpicture: y
		Then, in the source file for an imagefile asset, you specify the value for the urlpicture field like this:
		<urlpicture>*relative_path_to/ filename.jpg*
		</urlpicture>
		Note that you must specify the location of the file with a relative path (relative to the directory in which you are running the XMLPost utility).

42.2.4 Sample XMLPost Configuration File

Here is a sample configuration file named imagefile.ini, used to import imagefile assets for a site named Fiscal News.

```
xmlpost.xmlfilenamefilter: xml
#xmlpost.xmlproxypost: Future
#xmlpost.xmlproxyport: 80
xmlpost.url: http://localhost/servlet/ContentServer
xmlpost.numargs: 6
xmlpost.argname1: pagename
xmlpost.argvalue1: OpenMarket/Xcelerate/Actions/RemoteContentPost
xmlpost.argname2: AssetType
xmlpost.argvalue2: ImageFile
xmlpost.argname3: authusername
xmlpost.argvalue3: user_author
xmlpost.argname4: authpassword
xmlpost.argvalue4: user
xmlpost.argname5: inifile
xmlpost.argvalue5: futuretense.ini
xmlpost.argname6: publication
xmlpost.argvalue6: FiscalNews

xmlpost.success: Success
xmlpost.failure: Error
xmlpost.logfile: ImageFilePost.txt

postpublication: y
postprimarypubid: y
```

```
postpublist: y

postcategory: y
truncategory: 4

postpath: y
truncpath: 255

postname: y
truncname: 32

posttemplate: y
trunctemplate: 32

postsubtype: y
truncsubtype: 24

postfilename: y
truncfilename: 64

poststartdate: y

postdescription: y
truncdescription: 128

postsource: y

posturlpicture: y
fileurlpicture: y

posturlthumbnail: y
fileurlthumbnail: y

postmimetype: y
postwidth: y
postheight: y
postalign: y
postalttext: y

postkeywords: y
multikeywords: combine
trunckeywords: 128

postimagedate: y
```

42.3 Using XMLPost Source Files

Source files must be made up of well-formed XML without the need for a document
type definition (DTD) file. Actually, the configuration file functions something like a
DTD file—it defines the tags that are processed in the source files.

The data in your source files must be tagged with tags whose names match the column
names for the table that holds assets of that type. For example, a source file for an
`imagefile` asset uses tags named `name`, `caption`, `picutureurl`, and so on.

This section describes what needs to be in your source files and what XMLPost does
with them. It does not describe how to automate the generation of your XML source
files. How you create your source files depends on the source of your data and the
tools that you have to convert your data into XML files.

This section includes the following topics:

42.3.1 Sample XMLPost Source File

Here is a sample source file for an `imagefile` asset. Its tags are defined in the sample configuration file in Sample XMLPost Configuration File.

```
<document>
<name>High Five 25</name>
<keyword>Five</keyword>
<category>a</category>
<artist>by Ann. Artist</artist>
<alttext>Congratulations</alttext>
<align>CENTER</align>
<caption>A man extends <keyword>congratulations</keyword> with a boy.</caption>
<pictureurl>/images/eZine/highfive.jpg</pictureurl>
</document>
```

How the Data is Passed (Posted)

All of the text contained between a pair of XML tags in a source file is passed to the `RemoteContentPost` element from XMLPost as a variable that uses the `Variables.`*tagname* syntax convention.

For example, this line of code:

```
<name>High Five 25</name>
```

is sent to `RemoteContentPost` as `Variables.name` and the value of `name` is the string `High Five`.

42.3.2 XMLPost and File Encoding

When the source file data doesn't use the WebCenter Sites system's default file encoding but the database can accommodate that character set, specify the alternate file encoding in the XML version statement at the beginning of the file. For example:

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

42.4 Using the XMLPost Utility

You can invoke the XMLPost utility from the command line, or from a script or batch file, or from a program.

No matter how you start XMLPost, you must provide the following information:

- The name of the configuration file to use

- The source files, which can be specified as a single file, a list of files, or a directory of files

See these topics:

- Before You Begin

- Running XMLPost from the Command Line

- Identifying Source Files

- Running XMLPost as a Batch Process

- Running XMLPost Programmatically

42.4.1 Before You Begin

- Before you can use the XMLPost utility, the following must be true:

  - Your asset types are created. (Otherwise, there are no database tables to import the assets into.)

  - Your content management sites are created and the appropriate asset types are enabled for each site.

  - If you are using workflow, your workflow processes are created.

  - Your **Start Menu** shortcuts are created and, if you are using workflow, they assign the appropriate workflow process to the appropriate asset types.

  - The templates for the asset type are created.

  - The association fields for the asset types are created. However, custom code is required to set the value of an association field using XMLPost. See Customizing RemoteContentPost and PreUpdate.

- When invoking XMLPost, include the following command before the classpath to ensure UTF-8 encoding: `-Dfile.encoding=UTF-8`.

  Note:

  - Check all `.jar` files for version number, which can differ from one WebCenter Sites patch to the next.

  - When creating a file encoded with UTF-8 for XML Post imports, ensure that it is created without BOM because the use of a BOM is neither required nor recommended for UTF-8. However, the use of BOM may be encountered where UTF-8 data is converted from other encoding forms that use BOM, or where BOM is used as a UTF-8 signature.

42.4.2 Running XMLPost from the Command Line

To run XMLPost:

1. Place the configuration file and source files in a directory on a system that has WebCenter Sites installed.

2. To get XMLPost up and running, set the following directories to classpath: `<ORACLE_HOME>\wcsites\webcentersites\sites-home\lib\*` and `<ORACLE_HOME>\oracle_common\modules\clients\*`.

3. Run the following command (on a single command line) from that directory.

   This example uses Windows syntax, for UNIX-based systems including Linux and Solaris, the forward-slash (/) and colon (:) should be used as separators:

> **Note:**
>
> You are no longer allowed to give user credentials in the `ini` file. Only way you can provide them now is via command line.

```
java -Xmx512m -Dfile.encoding=UTF-8 -Dhttpclient=true -classpath
<ORACLE_HOME>\wcsites\webcentersites\sites-home\lib\*;
<ORACLE_HOME>\oracle_common\modules\clients\*;
<cs_app_dir>\WEB-INF\lib\servlet-api.jar;
<Sites WLS Domain>\wcsites\wcsites\config\*
COM.FutureTense.XML.Post.XMLPostMain -sSourcefile.xml -cConfigfile.ini
-ufwadmin -pxceladmin
```

where:

- `-Xmx512m` sets the maximum memory to use, which is needed with larger inserts.

- `<cs_app_dir>` is the directory on your application server where the WebCenter Sites application has been deployed.

Note that there are several options for designating the source file. See Identifying Source Files for information.

> **Note:**
>
> The `j2ee.jar` file is part of the J2EE SDK. You must install the SDK before running XMLPost.
>
> In the command line, provide the path to the source files and configuration file that are not in the working directory. For example: `-s/products/product.xml`.

42.4.3 Identifying Source Files

The source parameter that you use to identify the source files to the XMLPost utility can point to any of the following:

- A single file.

- A directory of files. All the files in that directory that have the file extension (typically .xml) designated by the configuration file will be posted (imported).

- A list file that provides a list of all the files that you want to import. It is similar to an .ini file but it has a file extension of .lst.

42.4.3.1 A Single File

To post the contents of one file, specify the name of that file in the command line. The following example instructs XMLPost to use a configuration file named `articlepost.ini` and one source file named `article.xml`.

This example uses Windows syntax, for UNIX-based systems including Linux and Solaris, the forward-slash (/) and colon (:) should be used as separators:

```
java -Xmx512m -Dfile.encoding=UTF-8 -classpath
<cs_app_dir>\WEB-INF\lib\apache-mime4j-0.5.jar;
```

```
<cs_app_dir>\WEB-INF\lib\commons-lang-2.5.jar;
<cs_app_dir>\WEB-INF\lib\commons-lang3-3.1.jar;
<cs_app_dir>\WEB-INF\lib\commons-codec-1.7.jar;
<cs_app_dir>\WEB-INF\lib\commons-logging-1.1.3.jar;
<cs_app_dir>\WEB-INF\lib\cs.jar;
<cs_app_dir>\WEB-INF\lib\cs-core.jar;
<cs_app_dir>\WEB-INF\lib\httpclient-4.3.6.jar;
<cs_app_dir>\WEB-INF\lib\http_client-1.0.0-150521.1418.jar;
<cs_app_dir>\WEB-INF\lib\httpcore-4.3.3.jar;
<cs_app_dir>\WEB-INF\lib\httpmime-4.3.6-12.1.4.0.1-SNAPSHOT.jar;
<cs_app_dir>\WEB-INF\lib\MSXML.jar;
<cs_app_dir>\WEB-INF\lib\servlet-api.jar;
<cs_app_dir>\WEB-INF\lib\spring-core-3.2.6.RELEASE.jar;
<cs_app_dir>\WEB-INF\lib\sites-security.jar;
<cs_app_dir>\WEB-INF\lib\esapi-2.0.1.jar;
<cs_app_dir>\WEB-INF\lib\ojdl-1.1.0-150521.1418.jar;<j2eeSDK_dir>\j2ee.jar;
<cs_app_dir>\WEB-INF\classes
 COM.FutureTense.XML.Post.XMLPostMain-sarticle.xml -carticlepost.ini
```

42.4.3.2 A Directory of Files

To post all the files in a directory, specify the path to that directory in the command
line. The following example instructs XMLPost to import the files in the
`xmlpostfiles` directory.

This example uses Windows syntax, for UNIX-based systems including Linux and
Solaris, the forward-slash (/) and colon (:) should be used as separators:

```
java -Xmx512m -Dfile.encoding=UTF-8 -classpath
<cs_app_dir>\WEB-INF\lib\apache-mime4j-0.5.jar;
<cs_app_dir>\WEB-INF\lib\commons-lang-2.5.jar;
<cs_app_dir>\WEB-INF\lib\commons-lang3-3.1.jar;
<cs_app_dir>\WEB-INF\lib\commons-codec-1.7.jar;
<cs_app_dir>\WEB-INF\lib\commons-logging-1.1.3.jar;
<cs_app_dir>\WEB-INF\lib\cs.jar;
<cs_app_dir>\WEB-INF\lib\cs-core.jar;
<cs_app_dir>\WEB-INF\lib\httpclient-4.3.6.jar;
<cs_app_dir>\WEB-INF\lib\http_client-1.0.0-150521.1418.jar;
<cs_app_dir>\WEB-INF\lib\httpcore-4.3.3.jar;
<cs_app_dir>\WEB-INF\lib\httpmime-4.3.6-12.1.4.0.1-SNAPSHOT.jar;
<cs_app_dir>\WEB-INF\lib\MSXML.jar;
<cs_app_dir>\WEB-INF\lib\servlet-api.jar;
<cs_app_dir>\WEB-INF\lib\spring-core-3.2.6.RELEASE.jar;
<cs_app_dir>\WEB-INF\lib\sites-security.jar;
<cs_app_dir>\WEB-INF\lib\esapi-2.0.1.jar;
<cs_app_dir>\WEB-INF\lib\ojdl-1.1.0-150521.1418.jar;<j2eeSDK_dir>\j2ee.jar;
<cs_app_dir>\WEB-INF\classes
 COM.FutureTense.XML.Post.XMLPostMain-sxmlpostfiles -carticlepost.ini
```

42.4.3.3 A List File

As an alternative to specifying a directory, you can create a list file that uses the format
of an `.ini` file and includes the following properties:

- `numfiles`, which specifies how many files are included in the list.

- `fileN`, which specifies the path to a file and its file name. The `N` stands for the
  file's order in the list file. The first file listed is `file1`, the second is `file2`, and so
  on.

  The value of `N` for the last `fileN` in the list must match the value specified by the
  `numfiles` property. XMLPost stops importing when it has imported as many

files as it is told to expect by the `numfiles` property. XMLPost does not import more files than `numfiles` states.

The file extension for a list file must be `.lst`.

The following sample list file is named `xmlpostfiles.lst`:

```
numfiles: 3
file1: c:\xmlpost\article1.xml
file2: c:\xmlpost\article2.xml
file3: c:\xmlpost\article3.xml
```

To post the files referenced in this file list, specify the name of the list file in the command line. The following example instructs XMLPost to import the files specified in the `xmlpostfiles.lst` file.

This example uses Windows syntax, for UNIX-based systems including Linux and Solaris, the forward-slash (/) and colon (:) should be used as separators:

```
java -Xmx512m -Dfile.encoding=UTF-8 -classpath
<cs_app_dir>\WEB-INF\lib\apache-mime4j-0.5.jar;
<cs_app_dir>\WEB-INF\lib\commons-lang-2.5.jar;
<cs_app_dir>\WEB-INF\lib\commons-lang3-3.1.jar;
<cs_app_dir>\WEB-INF\lib\commons-codec-1.7.jar;
<cs_app_dir>\WEB-INF\lib\commons-logging-1.1.3.jar;
<cs_app_dir>\WEB-INF\lib\cs.jar;
<cs_app_dir>\WEB-INF\lib\cs-core.jar;
<cs_app_dir>\WEB-INF\lib\httpclient-4.3.6.jar;
<cs_app_dir>\WEB-INF\lib\http_client-1.0.0-150521.1418.jar;
<cs_app_dir>\WEB-INF\lib\httpcore-4.3.3.jar;
<cs_app_dir>\WEB-INF\lib\httpmime-4.3.6-12.1.4.0.1-SNAPSHOT.jar;
<cs_app_dir>\WEB-INF\lib\MSXML.jar;
<cs_app_dir>\WEB-INF\lib\servlet-api.jar;
<cs_app_dir>\WEB-INF\lib\spring-core-3.2.6.RELEASE.jar;
<cs_app_dir>\WEB-INF\lib\sites-security.jar;
<cs_app_dir>\WEB-INF\lib\esapi-2.0.1.jar;
<cs_app_dir>\WEB-INF\lib\ojdl-1.1.0-150521.1418.jar;<j2eeSDK_dir>\j2ee.jar;
<cs_app_dir>\WEB-INF\classes
 COM.FutureTense.XML.Post.XMLPostMain-sc:\xmlpostfiles.lst -carticlepost.ini
```

42.4.4 Running XMLPost as a Batch Process

When you import assets of multiple types, identify a unique configuration file for each asset type by running XMLPost individually for each type. You can run XMLPost either manually, or automatically from a batch file. In the batch file, include a command line statement for each asset type (the statement identifies the configuration file and the location of the source files). You can use any of the ways described in the preceding section to identify the source files.

42.4.5 Running XMLPost Programmatically

You can also invoke the XMLPost utility programmatically by creating an XMLPost object and calling the `doIt` method `doIt(String[] args)`, where the input is a string array. The elements of the array are the same flags that you use when running XMLPost from the command line.

For example:

```
String args [] = {"-sSourcefile.xml","-cConfigfile.ini"};
COM.FutureTense.XML.Post.XMLPost poster = new
COM.FutureTense.XML.Post.XMLPost();
```

```
try {
  poster.doIt(args);
  } catch (Exception e) {
  e.printStackTrace();("error in XMLPost under program control");
}
```

Note that you must include the complete path to source files and configuration file.

42.5 Customizing RemoteContentPost and PreUpdate

If necessary, you can customize the XMLPost process by adding or modifying code in the `RemoteContentPost` element or the `PreUpdate` element for your asset types.

To import information about an asset to other tables, you must modify the `PreUpdate` element for that asset type.

This section provides two customization examples:

- Customizing the `PreUpdate` element for the `Article` asset type so that it sets headline information in the description field. There is a description column in the `Article` table but the field in the New or Edit article form is called **Headline**.

- Customizing the `PreUpdate` element for the `Article` asset type so that it can add associations to articles.

See these topics:

- Setting a Field Value Programmatically

- Setting an Asset Association

42.5.1 Setting a Field Value Programmatically

The article asset type has a field in the New and Edit forms called **Headline**, whose value is stored in the `description` column in the `Article` table. In order for headline text to be written to the correct column, when an `Article` asset is imported (that is, the `description` column), the `PreUpdate` element for the `Article` asset type is modified.

First, examine the sample configuration file named `ArticlePost.ini` that is located in the `Xcelerate/Samples/XMLPost` directory in your WebCenter Sites product kit. It has a tag specified for the **Headline** field:

```
# headline gets stored in the description field
postheadline: y
```

The following code in the `PreUpdate` element for the Article asset type writes the data that `RemoteContentPost` passes in as `Variable.headline` to the correct database column:

```
<if COND="IsVariable.headline=true">
  <then>
    <ASSET.SET NAME="theCurrentAsset"
     FIELD="description"
     VALUE="Variables.headline"/>
  </then>
</if>
```

This example uses a tag called `ASSET.SET`. This tag sets data in a field for the asset that is currently in memory. It takes three parameters:

- NAME: (required). The name of the asset object that is in memory. This asset object must have been previously instantiated either with the ASSET.LOAD tag or the ASSET.CREATE tag. By convention, WebCenter Sites uses the name theCurrentAsset to refer to the current asset object.

- FIELD: (required). The name of the field whose value you want to set. The name of this field must exactly match the name of a column in the storage table for assets of this type.

- VALUE: (required). The data to be inserted in the column.

42.5.2 Setting an Asset Association

The information about association between assets is written to the AssetRelationTree table. Because the standard behavior of XMLPost is to write asset information to the primary storage table of the asset type only, you must modify the PreUpdate element for the asset type to specify asset associations. For example, the Article asset type has an association field named MainImageFile. When a content provider creates an article asset, she selects the appropriate imagefile asset in this field.

Examine the sample configuration file named ArticlePost.ini that is located in the Xcelerate/Samples/XMLPost directory in your WebCenter Sites product kit. It has a tag specified for the MainImageFile association field:

```
postMainImageFile-name: y
```

The following code in the PreUpdate element for the article asset type writes the data that RemoteContentPost passes in as Variable.mainimagefile to the correct database table:

```
<if COND="IsVariable.MainImageFile-name=true">
<then>
 <ASSET.LOAD NAME="anAssociatedImage" TYPE="ImageFile"
  FIELD="name" VALUE="Variables.MainImageFile-name"/>
  <if COND="IsError.Variables.errno=false">
    <then>
      <ASSET.GET NAME="anAssociatedImage" FIELD="id" OUTPUT="imageid"/>
      <ASSET.ADDCHILD NAME="theCurrentAsset" TYPE="ImageFile"
       CHILDID="Variables.imageid" CODE="MainImageFile"/>
    </then>
  </if>
</then>
</if>
```

Note:

The ASSET.ADDCHILD tag creates only the link between the two assets. It does not create the associated asset. In order for this code to work, the asset specified with the CHILDID parameter must exist in the WebCenter Sites database.

This example uses a tag named ASSET.ADDCHILD. This tag associates a child asset with the asset that is currently held in memory. It takes five parameters:

- NAME (required). The name of the asset object that is in memory. This asset object must have been previously instantiated either with the ASSET.LOAD tag or the

ASSET.CREATE tag. By convention, WebCenter Sites uses the name theCurrentAsset to refer to the current asset object.

- TYPE (required). The asset type of the child asset.

- CHILDID (required). The ID of the child asset.

- CODE (optional). The name of the association. This value is written to the ncode column in the AssetRelationTree table.

- RANK (optional). A numeric value to establish an order for the child assets. This value is written to the nrank column in the AssetRelationTree table.

For information about ASSET.GET and ASSET.LOAD, see the *Oracle Fusion Middleware Tag Reference for Oracle WebCenter Sites Reference*.

42.6 Troubleshooting XMLPost

Sometimes XMLPost doesn't run, and it doesn't create a log file. Sometimes, when you're trying to save an asset, the error 105 is triggered. Read further to know more about XMLPost errors and debugging the Posting element.

This is a brief list of some possible problems that can occur when you run the XMLPost utility.

42.6.1 XMLPost Does Not Run and Does Not Create a Log File Message

There are two possible reasons for XMLPost to not start:

- An invalid server name has been specified in the xmlpost.URL property setting in your configuration file.

- WebCenter Sites is not running on the system you are importing to. Start it.

42.6.2 XMLPost Fails and there is a Missing Entity Statement in the Log File

This message means that there is invalid XML in the source file. Typically, your XML includes HTML code and that code includes special HTML characters that are not referred to by their character entity codes. For best coding practice, embed any HTML code in a <![CDATA[...]]> tag.

42.6.3 Error 105 is Triggered when XMLPost Tries to Save an Asset

There are several reasons why saving an asset can cause a database error. One common reason for a 105 error is XMLPost trying to save data that is too large for the column (field). Resolving this depends on your goals. If it is acceptable for XMLPost to truncate the data that doesn't fit into the column, you can add a trunctag property to the configuration file. For example, truncbody: 2000.

Another common reason for this error code is that an asset of that type with the same name exists. Try changing the name of the asset and importing the asset again.

42.6.4 Debugging the Posting Element

Use the XML Debugger utility to test the RemoteContentPost element if you have modified it or created your own posting element. To use XML Debugger, replace ContentServer with DebugServer in the xmlpost.url property setting. For example, change xmlpost.url: http://6ipjk/servlet/ContentServer to xmlpost.url: http://6ipjk/servlet/DebugServer

See XML Debugger utility in Introducing WebCenter Sites Tools and Utilities.

43

Importing Flex Assets

WebCenter Sites provides the XMLPost utility and a bulk processing utility named BulkLoader to import flex assets.

For information about importing flex assets using the XMLPost utility, see these topics:

- About Importing Flex Assets
- Understanding XMLPost and the Flex Asset Model
- About Importing the Structural Asset Types in the Flex Model
- Importing Flex Assets with XMLPost
- Editing Flex Assets with XMLPost
- Deleting Assets with XMLPost

For an in-depth information about using the BulkLoader utility, see Importing Flex Assets with the BulkLoader Utility.

43.1 About Importing Flex Assets

WebCenter Sites provides two utilities for importing assets that use the flex data model into the WebCenter Sites database, XMLPost and BulkLoader. XMLPost. WebCenter Sites provides three additional posting elements that work with XMLPost: `addData`, `modifyData`, and `deleteData`

See these topics:

- Before You Begin Importing the Data Structure Flex Asset Types
- About Importing the Flex Assets
- Overview of the Process to Import Flex Assets
- About Custom Data Delimiters

43.1.1 Before You Begin Importing the Data Structure Flex Asset Types

Before you can use either method, you must first create or import the data design or structural asset types into your flex families with XMLPost and the standard posting element, `RemoteContentPost`, provided by the WebCenter Sites product. That is, first you create or import the attribute editors, flex attributes, flex definitions, and flex parent definitions with the standard XMLPost posting element.

To use the BulkLoader utility, the flex parents must also be imported with XMLPost or created.

43.1.2 About Importing the Flex Assets

After importing your data structure asset types, import your flex assets using one of the two import methods:

- Use BulkLoader to import a large number (thousands or hundreds of thousands) of flex assets.

- Use the posting element to load a moderate number (hundreds) of flex and flex parent assets.

43.1.2.1 When to Use BulkLoader

When working within the basic asset model, it is typical to use XMLPost to import assets into the database on the management system and then publish those assets to the delivery system. This methodology changes with flex assets because the volume of data involved in a flex asset data model tends to be much greater than that in a basic asset model.

You use the BulkLoader utility during the initial setup of your WebCenter Sites system. See Importing Flex Assets with the BulkLoader Utility.

43.1.2.2 When to Use XMLPost

For regular or incremental updates after the initial setup of your WebCenter Sites system, perhaps some or all of your data originates in an ERP system, for example, you use the XMLPost utility and the addData posting element.

43.1.3 Overview of the Process to Import Flex Assets

Because assets using the flex model have dependencies on each other, flex asset types must be imported in a specific sequence. And, as with basic assets, the asset types must exist, sites must be created, and so on before you use XMLPost to import assets.

For information about the basic prerequisites for using XMLPost that apply to all asset types (both asset models), see Before You Begin.

After those basic requirements are met, you must import your flex asset types into the WebCenter Sites database on the management system in the following sequence:

1. Attribute editors are optional. To use attribute editors, either import them or create them before you import your flex attributes. The configuration file must instruct XMLPost to call the RemoteContentPost element. See Attribute Editors.

2. Flex attributes. The configuration file must instruct XMLPost to call the RemoteContentPost element. See Flex Attributes.

3. Flex parent definitions. The configuration file must instruct XMLPost to call the RemoteContentPost element. See Flex Definitions and Flex Parent Definitions: Sample Files.

> **Note:**
>
> You must import the flex parent definitions in the proper order. A parent definition asset referred by another parent definition must exist in the database.
>
> It is typical to import parent definitions one hierarchical level at a time, starting with the top level definitions.

4. Flex definitions. The configuration file must instruct XMLPost to call the `RemoteContentPost` element. See Flex Definitions and Flex Parent Definitions: Sample Files.

5. Flex parent assets. Do one of the following:

   - Import the flex parents individually or as part of the flex family tree for a flex assets using XMLPost.

   - To import the flex assets using the BulkLoader utility, first import the flex parent assets using XMLPost. The configuration file must instruct XMLPost to call the `RemoteContentPost` element. The file cannot specify the `addData` element because you are importing the parents without the entire family tree for the flex assets.

   See Flex Parents.

6. (Optional) First approve and publish all of the structural assets (attribute editors, flex attributes, flex definitions, parent definitions, and flex parents) from the management system to the delivery system, and then import flex assets into both the systems using the BulkLoader utility.

7. Flex assets. Do one of the following:

   - Use the BulkLoader utility. See Importing Flex Assets with the BulkLoader Utility.

   - Use XMLPost. See Importing Flex Assets with XMLPost.

You must follow the sequence outlined in the preceding steps because there are dependencies built in to the data structure of a flex asset family. Additionally, note the following dependencies:

- If a flex parent or flex asset has an attribute of type `asset`, the asset that you designate as the value of that attribute field must have been created or imported.

- An asset that you set as the value for an attribute of type `asset` must be of the correct asset type.

- When you are using XMLPost to create an asset that has an asset attribute of Type `asset`, you must use the unique name of the asset for this attribute value. Non-unique value for this attribute is not supported.

43.1.4 About Custom Data Delimiters

Custom data delimiters are used when an out of the box delimiter is used in the content. If the data you are importing, editing, or deleting via XMLPost uses a different data delimiting schema than the WebCenter Sites default schema (see the

table below for CS-default delimiter characters), specify custom delimiters as explained in the following table.

Table 43-1   Custom Data Delimiters

Tag	Property	Description
`<_xmlnamevaldelim_>`	`post_xmlnamevaldelim_`	Optional.
		Lets you specify a custom character for delimiting name/value pairs from one another.
		To specify a custom delimiter:
		1. Set the property to `y` in the configuration file.
		2. Use the tag in your XML file to define the custom delimiter. For example, to use the "at" character as a delimiter:
		`<_xmlnamevaldelim_>@</_xmlnamevaldelim_>`
		The default delimiter is the colon (`:`).
`<_xmlpostequaldelim_>`	`post_xmlpostequaldelim_`	Optional.
		Lets you specify a custom character for delimiting attribute names from their values.
		To specify a custom delimiter:
		1. Set the property to `y` in the configuration file.
		2. Use the tag in your XML file to define the custom delimiter. For example, to use two equal signs as a delimiter:
		`<_xmlpostequaldelim_>==</_xmlpostequaldelim_>`
		The default delimiter is the equal sign (`=`).
`<_xmlpostmulvaldelim_>`	`post_xmlpostmulvaldelim_`	Optional.
		Lets you specify a custom character for delimiting the values of a multivalued attribute from one another.
		To specify a custom delimiter:
		1. Set the property to `y` in the configuration file.
		2. Use the tag in your XML file to define the custom delimiter. For example, to use a hyphen as a delimiter:
		`<_xmlpostmulvaldelim_>-</_xmlpostmulvaldelim_>`
		The default delimiter is the semicolon (`;`).

43.2 Understanding XMLPost and the Flex Asset Model

The XMLPost utility works the same no matter which asset model or WebCenter Sites product you are using. However, for flex assets that store their data in multiple database tables, WebCenter Sites provides additional processing logic in some standard elements. This processing logic enables the flex asset types to support XMLPost.

Additionally, WebCenter Sites provides both a posting element that enables you to use XMLPost to edit flex assets (`modifyData`) and a posting element that enables you to use XMLPost to delete assets of any type (`deleteData`).

This chapter provides additional information about creating configuration and source files specifically for the asset types in a flex family (and attribute editors). Be sure to

also read Importing Assets of Any Type for basic information that pertains to all XMLPost configuration and source files.

In the flex asset model, you specify a different posting element based on the following categories of asset types:

- Structural asset types that give the flex asset type and flex parent asset type their data structure. That is, attribute editors, attributes, flex definitions, and flex parent definitions.

    Use the standard WebCenter Sites posting element `RemoteContentPost` to import the structural asset types. (You cannot use the `addData` element with assets of these types.)

- Flex and flex parent asset type (for example, product and product parent types).

    Depending on the situation, either use the posting element `addData` to import the flex and flex parent asset types or the posting element `RemoteContentPost`. (See Flex Parents and Importing Flex Assets with XMLPost for information about which posting element to use.)

In both cases, you create configuration files and source files (as described in About Importing Flex Assets and supplemented in this chapter), and then invoke the XMLPost utility (as described in Using the XMLPost Utility).

> **Note:**
>
> For reference, sample XMLPost code is provided **in** the WebCenter Sites installer package, in the `/Xcelerate/Samples/XMLPost` directory. The same folder contains the `readme.txt` file that describes the sample files.

Internal Names vs. External Names

When you create your flex family of asset types (see Creating a Flex Family in Creating a Flex Asset Family), you specify both an internal and an external name for your asset types.

The internal name is used for the primary storage table in the database. The external name is used in the New, Edit, and Inspect forms, in search results list, and so on. For example, the internal name for the attribute editor asset type is `AttrTypes`, but that name is not used in the user interface.

Because XMLPost communicates with the database, you must always use the internal name of the asset type in the configuration files and source files. For example, in a configuration file for attribute editors, you would specify the following:

```
postargname2: AssetType
postargvalue2: AttrTypes
```

43.3 About Importing the Structural Asset Types in the Flex Model

The configuration and source files for the flex asset types are similar to those for basic assets. Let's take a look at sample configuration and source files for the structural flex asset types.

For information about configuration and source files, see Importing Assets of Any Type.

See these topics:

- [Attribute Editors](#)
- [Flex Attributes](#)
- [Flex Definitions and Flex Parent Definitions: Sample Files](#)
- [Flex Parents](#)

43.3.1 Attribute Editors

Attribute editors store their data in one table, named `AttrTypes`. `AttrTypes` is the internal name of the attribute editor asset type. Be sure to use this name in your configuration file for attribute editors.

This table describes the configuration file properties and source file tags that you use with attribute editors:

Table 43-2   Attribute Editor Tag and Properties

Tag	Property	Description
`<name>`	`postname`	Required for all asset types.
		Name of the attribute editor asset. Attribute names are limited to 64 characters and cannot contain spaces.
`<description>`	`postdescription`	Optional.
		Description of the use or function of the attribute.
`<AttrTypeText>`	`postAttrTypeText`	Required.
		Either the name of the file with the attribute editor XML code, or the actual code.
		This tag corresponds to the XML in file field and Browse button and the XML field in the New and Edit attribute editor forms in the WebCenter Sites interface.

43.3.1.1 Sample Configuration File: Attribute Editor

This is a sample configuration file for the attribute editor asset type. It works with the sample source file immediately following this example.

```
xmlpost.xmlfilenamefilter: .xml
xmlpost.url: http://izod19/servlet/ContentServer
xmlpost.numargs: 6
xmlpost.argname1: pagename
xmlpost.argvalue1: OpenMarket/Xcelerate/Actions/RemoteContentPost
xmlpost.argname2: AssetType
xmlpost.argvalue2: AttrTypes
# notice that you use the internal name of the asset type

xmlpost.argname3: authusername
xmlpost.argvalue3: user_editor
xmlpost.argname4: authpassword
xmlpost.argvalue4: user
xmlpost.argname5: inifile
xmlpost.argvalue5: futuretense.ini
xmlpost.argname6: startmenu
xmlpost.argvalue6: New Attribute Editor

xmlpost.success: Success
xmlpost.failure: Error
```

```
xmlpost.logfile: attreditorpostlog.txt

xmlpost.deletefile: y

postpublication: y

postname: y
postdescription: y
postAttrTypeText: y
```

43.3.1.2 Sample Source File: Attribute Editor

The following source file is tagged for importing a check box attribute editor, or presentation object. It works with the preceding sample configuration file.

```
<document>
<publication>AA Illumination</publication>
<name>Editor4-CheckBoxes</name>
<description>Attribute Type Four Check Box</description>
<AttrTypeText>
  <![CDATA[
    <?XML VERSION="1.0"?>
    <!DOCTYPE PRESENTATIONOBJECT SYSTEM "presentationobject.dtd">
    <PRESENTATIONOBJECT NAME="CheckBoxTest">
      <CHECKBOXES LAYOUT="VERTICAL">
        <ITEM>Red</ITEM>
        <ITEM>Green</ITEM>
        <ITEM>Blue</ITEM>
      </CHECKBOXES>
    </PRESENTATIONOBJECT>
  ]]>
</AttrTypeText>
</document>
```

43.3.2 Flex Attributes

Flex attributes have several tables, but XMLPost writes to only two of them: the main storage table and the attribute asset type's `_Extension` table.

This means that the source file section of the configuration file must specify and the source file itself must use tags that represent columns in both tables. Those source file tags and configuration file properties are shown in the following table:

Table 43-3   Flex Attribute Tags and Properties

Tag	Property	Description
<name>	postname	Required for all asset types.
		Name of the attribute. Attribute names are limited to 64 characters and cannot contain spaces.
<description>	postdescription	Optional.
		Description of the use or function of the attribute.
<valuestyle>	postvaluestyle	Optional.
		Whether the attribute can hold a single value (S) or multiple values (M). If no, this tag is not used, the attribute is set to hold a single value by default.

Table 43-3 (Cont.) Flex Attribute Tags and Properties

Tag	Property	Description
`<type>`	`posttype`	Required. The data type of the attribute. Valid options are `asset, date, float, int, money, string, text,` or `blob`.
`<assettypename>`	`postassettypename`	Required if `<type>` is set to `asset`. The name of the asset type that the attribute holds.
`<upload>`	`postupload`	Required if `<type>` is set to `blob`. The path to the directory in which you want to store the attribute values. Note that the value that you enter in this field is appended to the value set as the default storage directory (defdir) for the attribute table by the `cc.urlattrpath` property in the `wcs_properties.json` file.
`<attributetype>`	`postattributetype`	Optional. The name of the attribute editor to use, if applicable.
`<enginename>`	`postenginename`	Optional. The name of the search engine you may be using on your management system.
`<charsetname>`	`postcharsetname`	Optional. The search engine character set to use. By default, it is set to ISO 8859-1.
`<editing>`	`postediting`	Foreign attributes only. Whether a foreign attribute can be edited through the WebCenter Sites forms (`L`), or edited externally using a third-party tool (`R`). `L` is the default.
`<storage>`	`poststorage`	Foreign attributes only. Whether the values for a foreign attribute are to be stored in a `_Mungo` table in the WebCenter Sites database (`L`) or in a foreign table (`R`). `L` is the default.
`<externalid>`	`postexternalid`	Foreign attributes only. The name of the column that serves as the primary key for the table that holds this foreign attribute; that is, the column that uniquely identifies the attribute.
`<externalcolumn>`	`postexternalcolumn`	Foreign attributes only. The name of the column in the foreign table that holds the values for this attribute.
`<externaltable>`	`postexternaltable`	Foreign attributes only. The name of the foreign table that contains the columns identified by externalid and external column.
`<publication>`	`postpublication`	Optional. The names of all the sites that can use this attribute.

43.3.2.1 Sample Configuration File: Flex Attribute

This is a sample configuration file for a product attribute asset type works with the sample source file immediately following this example:

```
xmlpost.xmlfilenamefilter: .xml

xmlpost.url: http://izod19/servlet/ContentServer
xmlpost.numargs: 6
xmlpost.argname1: pagename
xmlpost.argvalue1: OpenMarket/Xcelerate/Actions/RemoteContentPost
xmlpost.argname2: AssetType
xmlpost.argvalue2: PAttributes
# Notice that this is the internal name of the asset
# type. The external name of this asset type is
# Product Attribute.

xmlpost.argname3: authusername
xmlpost.argvalue3: user_editor
xmlpost.argname4: authpassword
xmlpost.argvalue4: user
xmlpost.argname5: inifile
xmlpost.argvalue5: futuretense.ini
xmlpost.argname6: startmenu
xmlpost.argvalue6: New Product Attribute

xmlpost.success: Success
xmlpost.failure: Error
xmlpost.logfile: attributespostlog.txt

xmlpost.deletefile: y

postpublication: y
postname: y
postattributetype: y
postdescription: y
postvaluestyle: y
posttype: y
postediting: y
poststorage: y
postenginename: y
poststatus: y
postassettypename: y
postupload: y
postexternalid: y
postexternalcolumn: y
postexternaltable: y
postcharsetname: y
```

43.3.2.2 Sample Source File: Attribute

This is a sample source file for importing a product attribute named footnotes. It works with the preceding sample configuration file.

```
<document>
  <publication>AA Illumination</publication>
  <name>footnotes</name>
  <description>Footnotes</description>
  <valuestyle>S</valuestyle>
  <type>URL</type>
  <editing>L</editing>
```

```
        <storage>L</storage>
</document>
```

> **Note:**
>
> Remember that all the dependencies and restrictions concerning the data type of a flex attribute apply whether you are creating an attribute through the WebCenter Sites interface (the **New** or **Edit** flex attribute forms) or through XMLPost. See Create Flex Attributes in Creating a Flex Asset Family.

43.3.3 Flex Definitions and Flex Parent Definitions: Sample Files

The flex definition and flex parent definition asset types are very similar and you code their configuration and source files in nearly the same way. They require several of the same tags in their source files and the same properties in their configuration files. Each has one additional property/tag.

This section includes the following topics:

- Sample Configuration File: Flex Definition

- Sample Source File: Flex Definition

The source file tags and configuration file properties for flex definitions and flex parent definitions are listed in the following table. Note that they are case-sensitive.

Table 43-4   Flex Definition and Flex Parent Definition Tags and Properties

Flex definition and flex parent definition tag and property	Description

tag:
`<internalname>`
property:
`postinternalname`
	Required.
	The name of the asset; this is a required value for all asset types. Flex definition and flex parent definition names are limited to 64 characters and they cannot contain spaces.

tag:
`<internaldescription>`
property:
`postinternaldescription`
	Optional.
	The description of the use or function of the asset.

tag:
`<renderid>`
property:
`postrenderid`
	Optional. For flex definitions only.
	The ID of the Template asset that is to be assigned to all the flex assets that are created with this flex definition.

tag:
`<parentselectstyle>`
property:
`postparentselectstyle`
	Optional. For flex parent definitions only.
	Defines how flex parents are to be selected when a user creates a flex asset using the definition.
	This property/tag represents the **Parent Select Style** field in the New and Edit parent definition forms.
	When using the tag in the source file, the options are `treepick` and `selectboxes`.

Table 43-4   (Cont.) Flex Definition and Flex Parent Definition Tags and Properties

Flex definition and flex parent definition tag and property	Description
The next four tags and properties perform the same function as the buttons and fields in the Product Parent Definition section on the New and Edit forms for parent definitions and flex definitions. See (Conditional) Creating Flex Filter Assets and Creating Flex Definition Assets in Creating a Flex Asset Family.	n/a

tag:
`<OptionalSingleParentList>`
property:
`postOptionalSingleParentList`
	Use this tag to specify any single optional parent definition.

tag:
`<RequiredSingleParentList>`
property:
`postRequiredSingleParentList`
	Use this tag to specify any single required parent definition.

tag:
`<RequiredMultipleParentList>`
property:
`postRequiredMultipleParentList`
	Use this tag to specify multiple required parent definitions.

tag:
`<OptionalMultipleParentList>`
property:
`postOptionalMultipleParentList`
	Use this tag to specify multiple optional parent definition.

The next three tags and properties perform the same functions as the buttons and fields in the Attributes section on the New and Edit forms for flex definitions and flex parent definitions. See (Conditional) Creating Flex Filter Assets and Creating Flex Definition Assets in Creating a Flex Asset Family.	n/a

tag:
`<RequiredAttrList>`
property:
`postRequiredAttrList`
	The list of attributes that are required for the flex parents or the flex assets that use the definition.

tag:
`<OptionalAttrList>`
property:
`postOptionalAttrList`
	The list of attributes that are optional for the flex parents or the flex assets that use the definition.

Table 43-4   (Cont.) Flex Definition and Flex Parent Definition Tags and Properties

Flex definition and flex parent definition tag and property	Description

tag:
`<OrderedAttrList>`
property:
`postOrderedAttrList`
	The order in which all attributes, be they required or optional, should appear in the New, Edit, Inspect, and similar forms.
	
	This tag replaces the other attribute tags. The example source file in this section shows an example of how to use this tag in a source file.

A configuration file must include all the properties that could be used by any one of the assets of the type that the configuration file works with. The individual source files include only the tags that are needed to define those individual assets.

43.3.3.1 Sample Configuration File: Flex Definition

The following example is a configuration file for importing product definitions. It works with the sample source file immediately following this example.

```
xmlpost.url: http://izod19/servlet/ContentServer
xmlpost.numargs: 6
xmlpost.argname1: pagename
xmlpost.argvalue1: OpenMarket/Xcelerate/Actions/RemoteContentPost
xmlpost.argname2: AssetType
xmlpost.argvalue2: ProductTmpls
# Notice that this is the internal name of the asset type.
# The external name of this asset type is
# Product Definition.

xmlpost.argname3: authusername
xmlpost.argvalue3: user_editor
xmlpost.argname4: authpassword
xmlpost.argvalue4: user
xmlpost.argname5: inifile
xmlpost.argvalue5: futuretense.ini
xmlpost.argname6: startmenu
xmlpost.argvalue6: New Product Definition

xmlpost.success: Success
xmlpost.failure: Error
xmlpost.logfile: productdefpostlog.txt
xmlpost.deletefile: y

postpublication: y
postinternalname: y
postinternaldescription: y

postparentselectstyle: y

postOptionalSingleParentList: y
postRequiredSingleParentList: y
postRequiredMultipleParentList: y
postOptionalMultipleParentList: y

postRequiredAttrList: y
postOptionalAttrList: y
postOrderedAttrList: y
```

```
postrenderid: y
```

43.3.3.2 Sample Source File: Flex Definition

The following source file, `lighting.xml`, is for a product definition named Lighting. It works with the preceding sample configuration file.

```
<document>
<publication>AA Illumination</publication>
<internalname>Lighting</internalname>
<internaldescription>Generic Lighting Template</internaldescription>
<RequiredAttrList>sku</RequiredAttrList>
  <OptionalAttrList>
    productdesc;caseqty;bulbshape;bulbsize;basetype;
    colortemp;meanlength;lightcenterlength;reducedwattage;beamspread;
    fixturetype;ballasttype;colorrenderingindex;minstarttemp;powerfactor;
    totalharmonicdist;spreadbeam10h;spreadbeam10v;spreadbeam50h;
    spreadbeam50v;halogen;operatingposition;filamenttype;bulbimage;
    baseimage;filamentimage;footnotes;price;life;voltage;wattage
  </OptionalAttrList>
<parentselectstyle>treepick</parentselectstyle>
<OptionalMultipleParentList>SubCategory</OptionalMultipleParentList>
</document>
```

Examine the preceding list of attributes. When you include multiple values in a tag, separate them from each other with a semicolon (;).

Note that while the optional/multiple parent model is used, there are other possible configurations:

```
<OptionalSingleParentList>flexparentdefinition</OptionalSingleParentList>
<RequiredSingleParentList>flexparentdefinition</RequiredSingleParentList>
<RequiredMultipleParentList>flexparentdefinition</RequiredMultipleParentList>
```

Supplying a List of Ordered Attributes

To use the `<OrderedAttrList>` tag because the attributes have to be displayed in a specific order, do not also include the `<RequiredAttrList>` and `<OptionalAttrList>` tags. In the string contained in the `<OrderedAttrList>` tag, specify which attributes are required and which are optional, as follows:

- For required attributes, precede the attribute name with R (required)

- For optional attributes, precede the attribute name with or O (optional)

- Be sure to list the attributes in order.

- Be sure to use a semicolon (;) to separate the values.

For example:

```
<OrderedAttrList>Rsku;Oproductdesc;Ocaseqty;Obulbshape;Obulbsize;Obasetype;Ocolortemp
;Omeanlength;Olightcenterlength;Oreducedwattage;<OrderedAttrList>
```

43.3.4 Flex Parents

You can use XMLPost to import flex parent assets in two ways:

- Individually. You code a separate XMLPost source file for each flex parent and an XMLPost configuration file that identifies the asset type and the pagename for the

standard `RemoteContentPost` posting element. To use the BulkLoader utility, first import the flex parent assets with XMLPost in this way.

- As part of the flex family tree for a flex asset. To import your flex assets (rather than the BulkLoader) using XMLPost, combine the flex parents with the flex assets and import the flex parents as a part of a flex family tree, within the context of a specific flex asset. You code a separate XMLPost source file for each flex asset and identify all the parents for that flex asset in that source file. XMLPost then creates the variables for one flex asset and multiple flex parents (if they do not yet exist) when it parses the source file.

This topic describes the source and configuration file for importing them individually. For information about importing them with the flex assets, see Importing Flex Assets with XMLPost.

This topic includes the following topics:

- Sample Configuration File: Individual Flex Parent

- Sample Source File: Individual Flex Parent

43.3.4.1 Sample Configuration File: Individual Flex Parent

The following example is a configuration file for importing product parents. It works with the sample source file immediately following this example.

```
xmlpost.xmlfilenamefilter: .xml

xmlpost.url: http://izod19/servlet/ContentServer
xmlpost.numargs: 6
xmlpost.argname1: pagename
xmlpost.argvalue1: OpenMarket/Xcelerate/Actions/RemoteContentPost
xmlpost.argname2: AssetType
xmlpost.argvalue2: ProductGroups
# notice that you use the internal name of the asset type

xmlpost.argname3: authusername
xmlpost.argvalue3: user_editor
xmlpost.argname4: authpassword
xmlpost.argvalue4: user
xmlpost.argname5: inifile
xmlpost.argvalue5: futuretense.ini
xmlpost.argname6: startmenu
xmlpost.argvalue6: New Product Parent

xmlpost.success: Success
xmlpost.failure: Error
xmlpost.logfile: productdefpostlog.txt
xmlpost.deletefile: y

postpublication: y
postinternalname: y
postinternaldescription: y
postflexgrouptemplateid: y
postfgrouptemplatename: y
postParentList: y
postcat1: y
postcat2: y
```

43.3.4.2 Sample Source File: Individual Flex Parent

The following source file creates a product parent (flex parent) named Halogen. It works with the preceding sample configuration file.

```
<document>
<publication>AA Illumination</publication>
<internalname>Halogen</internalname>
<fgrouptemplatename>Category</fgrouptemplatename>
<cat1>Halogen</cat1>
</document>
```

Remember that when you use the `RemoteContentPost` posting element, you must provide one source file for each parent asset.

43.4 Importing Flex Assets with XMLPost

Remember to import the structural asset types (attributes, flex definitions, and flex parent definitions) before you import flex assets with XMLPost.

Use `RemoteContentPost` or `addData` posting elements for flex assets:

- The `addData` posting element creates parent assets for the flex asset if they do not yet exist. For example, use this posting element for the initial import of your flex assets if you are not using the BulkLoader utility.

  When you use the `addData` posting element, the source file must specify the entire family tree for the flex asset. Each flex asset requires a separate source file, but you can specify any number of parents for that flex asset in that source file and XMLPost creates the flex asset and its parents (if they do not yet exist).

- The `RemoteContentPost` element creates flex assets and sets values for their parents. Those parents must exist. For example, use this posting element if you wish to perform the initial import using BulkLoader and then on use XMLPost.

  When you use `RemoteContentPost` to import a flex asset, the source file must specify only the asset's immediate parents (which requires you to include fewer lines of code). However, to create a new flex parent for the new flex asset, use the `addData` posting element and specify the entire family tree in the source file.

This section includes the following topics:

- Configuration File Properties and Source File Tags for Flex Assets

- Sample Flex Asset Configuration File for addData

- Configuration File Properties and Attributes of Type Blob (or URL)

- Sample Flex Asset Source File for addData

- Sample Flex Asset Configuration File for RemoteContentPost

- Sample Flex Asset Source File for RemoteContentPost

43.4.1 Configuration File Properties and Source File Tags for Flex Assets

As with the structural asset types, you must use the internal name of the flex and flex parent asset types in your configuration and source files. However, unlike the structural asset types, you do not have to include an argument for the asset type in the

configuration file. Source files for flex assets have a required tag that identifies the asset type so you do not have to repeat this information in the configuration file.

This section includes the following topics:

- For the addData Posting Element

- For the RemoteContentPost Posting Element

- For the RemoteContentPost Posting Element

43.4.1.1 For the addData Posting Element

The following table lists the source file tags and configuration file properties for flex assets (and their flex parents) when you are using the `addData` posting element. Note that they are case sensitive.

Table 43-5   addData Posting Element

Tag	Property	Description
`<_ASSET_>`	`post_ASSET_`	Required. The internal name of the asset type. For example, `Products`, `AArticles`, and `AImages`.
`<_TYPE_>`	`post_TYPE_`	Required. The name of the flex definition that this flex asset is using.
`<_ITEMNAME_>`	`post_ITEMNAME_`	Required. The name of the asset.
`<_ITEMDESCRIPTION_>`	`post_ITEMDESCRIPTIO N_`	Optional. The description of the asset.

`<_GROUP_parentDefin itionName>`	`post_GROUP_parentDe finitionName`	Optional.
		The flex asset's parents. The configuration file must include a tag for each possible parent definition. For example, if flex assets have parents that use either of two parent definitions named Division and Department, the following two properties to define a tag for each are required in the configuration file:
		`post_Group_Department`
		`post_Group_Division`

`<_GROUPDESCRIPTIONS _>`	`post_GROUPDESCRIPTI ONS_`	Optional. When designating a new parent include the description of the parent definition.
`<displaytype>`	`postdisplaytype`	Optional. The name of the `Template` asset for the flex asset.

Table 43-5   (Cont.) addData Posting Element

Tag	Property	Description
`<AttributeName>`	`postAttributeName`	Include a property in the configuration file for each attribute that assets of the type can have (both required and optional). The source files then have to supply a value for each required attribute and any optional ones that apply to that asset.
		For example, for an attribute named SKU, include a property called `postSKU` in the source files and lines of code like this:
		`<SKU>123445</SKU>`

43.4.1.2 For the RemoteContentPost Posting Element

The following table lists the source file tags and configuration file properties for flex assets (and their flex parents) when you are using the `RemoteContentPost` posting element. Note that they are case-sensitive.

Table 43-6   RemoteContentPost Posting Element

Tag	Property	Description
`<_DEFINITION_>`	`post_DEFINITION_`	Required. The name of the flex definition that this flex asset is using. (Note that `post_TYPE` will also work.)
`<_ITEMNAME_>`	`post_ITEMNAME_`	Required. The name of the asset.
`<_ITEMDESCRIPTION_>`	`post_ITEMDESCRIPTION_`	Optional. The description of the asset.
`<ParentList>`	`post_ParentList`	Optional. The flex asset's immediate parents.
`<template>`	`posttemplate`	Optional. The name of the Template asset for the flex asset. (Note that `postdisplaytype` will also work.)
`<AttributeName>`	`postAttributeName`	Include a property in the configuration file for each attribute that assets of the type can have (both required and optional). The source files then have to supply a value for each required attribute and any optional ones that apply to that asset.
		For example, for an attribute named SKU, include a property called `postSKU` in the source files and lines of code like this:
		`<SKU>123445</SKU>`

43.4.2 Sample Flex Asset Configuration File for addData

This is a sample configuration file for a product asset type. The file invokes the addData posting element and works with the source file example immediately following this example:

```
xmlpost.xmlfilenamefilter: .xml

#xmlpost.proxyhost: Future
#xmlpost.proxyport: 80

xmlpost.url: http://wally9:80/servlet/ContentServer

# notice that it uses addData
# rather than RemoteContentPost
xmlpost.numargs: 5

xmlpost.argname1: pagename
xmlpost.argvalue1: OpenMarket/Gator/XMLPost/addData

# Notice that you do not need to provide
# the name of the asset type because that information
# is required in the source files for flex assets.

xmlpost.argname2: inifile
xmlpost.argvalue2: futuretense.ini
xmlpost.argname3: authusername
xmlpost.argvalue3: editor
xmlpost.argname4: authpassword
xmlpost.argvalue4: xceleditor
xmlpost.argname5: startmenu
xmlpost.argvalue5: New Product

xmlpost.success: Success
xmlpost.failure: Error
xmlpost.logfile: productdatalog.txt

xmlpost.postdeletefile: y

post_ASSET_: y
post_ITEMNAME_: y
post_TYPE_: y
post_GROUP_Category: y
post_GROUP_SubCategory: y
postpublication: y
postsku: y
postproductdesc: y
postcaseqty: y
postbulbshape: y
postbulbsize: y
postbasetype: y
postcolortemp: y
postmeanlength: y
postlightcenterlength: y
postreducedwattage: y
postbeamspread: y
postfixturetype: y
postballasttype: y
postcolorrenderingindex: y
postminstarttemp: y
```

```
postpowerfactor: y
posttotalharmonicdist: y
postspreadbeam10h: y
postspreadbeam10v: y
postspreadbeam50h: y
postspreadbeam50v: y
posthalogen: y
postoperatingposition: y
postfilamenttype: y
postbulbimage: y
postbaseimage: y
postfilamentimage: y
postfootnotes: y
postcat1: y
postcat2: y
postprice: y
postvoltage: y
postwattage: y
postlife: y
```

43.4.3 Configuration File Properties and Attributes of Type Blob (or URL)

If the asset type has an attribute of type `blob` (or `url`), the configuration file needs two entries for the tag that references the attribute: one to identify the attribute and one to identify the file name of either the file that holds the content for the attribute (an upload field) or the name that you want WebCenter Sites to give the file that it creates from text entered directly into a text field (a text field of type `blob` or `URL`).

43.4.3.1 Attribute of Type Blob (or URL) As an Upload Field

An attribute of type `blob` can be an upload field. For example, a `blob` attribute named `footnotes` is an upload field with a **Browse** button for finding the file rather than a text field that you enter text into. Therefore, it has two properties:

- `posttag`, which in this scenario is `postfootnotes: y`

- `filetag`, which in this scenario is `filefootnotes: y`

When you include a value for this attribute in a source file, you use the following convention:

```
<footnotes>FileName.txt</footnotes>
```

Note that when you are importing an asset that has this kind of field (attribute), the file that holds the text that you want to store as the attribute value for the flex asset must be located in the same directory as the source file for the asset.

43.4.3.2 Attribute of Type Blob (or URL) As a Text Field

If the fictitious `footnotes` attribute is a field that takes text directly rather than a file, the configuration file requires the following properties:

- `postfootnotes: y`

- `postfootnotes_file: y`

Then, when you include a value for the attribute in the source file, you use the following convention:

```
<footnotes>lots and lots of text</footnotes>
<footnotes_file>FileNameYouWantUsed.txt</footnotes_file>
```

43.4.4 Sample Flex Asset Source File for addData

The following source file works with the example flex asset configuration file preceding this section.

This section includes the following topics:

- Sample File

- Handling Special Characters

- Flex Assets and Their Parents

- Specifying the Parents of a Flex Asset

- Setting Attribute Values for Parents

- Setting Multiple Values in a Flex Source File

43.4.4.1 Sample File

This source file creates a lightbulb product named `10004` from the product definition named `Lighting`:

```
<document>

# the first three tags are required
<_ASSET_>Products</_ASSET_>
<_ITEMNAME_>10004</_ITEMNAME_>
<_TYPE_>Lighting</_TYPE_>

# This tag is required because the publication is
# not set in the configuration file
<publication>AA Illumination</publication>

# This tag assigns a Template asset to the product
<displaytype>Lighting Detail</displaytype>

# The rest of these tags set flex attribute values for the product
<price>5</price>
<sku>10004</sku>
<productdesc>F4T5/CW</productdesc>
<caseqty>24</caseqty>
<bulbshape>T</bulbshape>
<bulbsize>5</bulbsize>
<basetype>Miniature Bipin (G5)</basetype>
<colortemp>4100</colortemp>
<meanlength></meanlength>
<lightcenterlength></lightcenterlength>
<reducedwattage></reducedwattage>
<beamspread></beamspread>
<fixturetype></fixturetype>
<ballasttype></ballasttype>
<colorrenderingindex>60</colorrenderingindex>
<minstarttemp></minstarttemp>
<powerfactor></powerfactor>
<totalharmonicdist></totalharmonicdist>
<spreadbeam10h></spreadbeam10h>
<spreadbeam10v></spreadbeam10v>
<spreadbeam50h></spreadbeam50h>
<spreadbeam50v></spreadbeam50v>
```

```
<halogen></halogen>
<operatingposition></operatingposition>
<filamenttype></filamenttype>
<bulbimage>BLB-260.gif</bulbimage>
<baseimage>BLB-250.gif</baseimage>
<filamentimage></filamentimage>
<footnotes>
</footnotes>
<life>6000</life>
<voltage></voltage>
<wattage>4</wattage>
<cat1>Fluorescent</cat1>
<cat2>Preheat Lamps</cat2>

<!-- GROUP tags that specify the parents. Remember that you have to
specify the entire family tree for the flex asset when using the addData posting
element-->

<_GROUP_Category>Fluorescent</_GROUP_Category>
<_GROUP_SubCategory>Preheat Lamps</_GROUP_SubCategory>
</document>
```

The preceding source file set the product's parent to Preheat Lamps and the parent of Preheat Lamps to Fluorescent.

43.4.4.2 Handling Special Characters

XMLPost uses the HTTP POST protocol, which means that it sends data in an HTTP stream. Therefore, certain characters are considered to be special characters and must be encoded because they are included in URLs.

In your source file, if any attribute values contains any special character, replace all its instances with its corresponding URL encoding sequence, found in About Values for Special Characters.

43.4.4.3 Flex Assets and Their Parents

The GROUP tags specify the parents in the family tree. When XMLPost uses the addData posting element and parses the GROUP section of the source file, it does the following:

1. Determines which parent definitions are legal for an asset using this flex definition.

2. For each legal parent definition, it verifies whether the source file specifies a parent of that definition:

   • If yes, it sets the parent. And if the parent does not yet exist, it creates the parent.

   • If no, it does not set the parent. However, if a parent of that definition is required, it returns an error.

43.4.4.4 Specifying the Parents of a Flex Asset

To specify the parents of a flex asset, you provide the name of the parents nested in the <_GROUP_parentDefinitionName> tag. For example:

```
<_GROUP_subcategory>Blacklights</_GROUP_subcategory>
```

Where `subcategory` is the name of the parent definition for the Blacklights parent (product parent).

Remember that you must specify the entire family tree for the flex asset. For example, in addition to specifying the parent for the lightbulb (Blacklights), you specify the grandparent:

```
<_GROUP_subcategory>Blacklights</_GROUP_subcategory>
```

```
<_GROUP_category>Fluorescent</_GROUP_category>
```

43.4.4.5 Setting Attribute Values for Parents

If an attribute can belong to multiple parents, in XMLPost specify to which parent the attribute belongs. For example, let's say that the `bulbshape` attribute is assigned to parents rather than products. In this case, you would include a line of code such as this:

```
<bulbshape>Halogen=T</bulbshape>
```

43.4.4.6 Setting Multiple Values in a Flex Source File

All of the tags that configure parents and the tags that specify attributes (while the attribute is configured to accept multiple values) can handle multiple values. Those tags are as follows:

- `_GROUP_parentDefinitionName`

- `_GROUPDESCRIPTIONS_`

- the attribute tags

When you have multiple parents from the same definition for a flex asset, you provide all of the names of the parents in the same `_GROUP_parentDefinitionName` tag and you use a semicolon (;) to separate the parent names.

For example:

```
<_GROUP_Cateogry>Incandescent;Halogen</_GROUP_Category>
```

When XMLPost imports this asset, it sets its parents as Incandescent and Halogen, which are both of the Category parent definition. If Incandescent and Halogen do not exist yet, XMLPost creates them.

Use a similar syntax to set multiple attribute values for the multiple parents. Once again, let's say that the Category definition requires that parents of that definition have a value for the bulbshape attribute. You can set the value of the bulbshape attribute for both of the parents that were specified by the `<_GROUP_Category>` tag as follows:

```
<bulbshape>Incandescent=E;K:Halogen=T</bulbshape>
```

Note the following about this syntax:

- You use `parentName=attributeValue` pairs to set the attribute value (`Halogen=T`).

- You use a colon to separate the parents from each other. (`Incandescent=S:Halogen=T`).

- You use a semicolon to separate the attribute values for a parent when that parent has multiple values for the attribute (`Incandescent=E;K:Halogen=T`).

And, as mentioned, specify descriptions for the parents that you identify in the same tag, too. For example:

```
<_GROUPDESCRIPTIONS>
Incandescent=From Detroit:Halogen=From Chicago
</_GROUPDESCRIPTIONS>
```

43.4.5 Sample Flex Asset Configuration File for RemoteContentPost

This is a sample configuration file for a product asset type. It works with the source file example immediately following this example.

```
xmlpost.xmlfilenamefilter: .xml

#xmlpost.proxyhost: Future
#xmlpost.proxyport: 80

xmlpost.url: http://wally9:80/servlet/ContentServer
xmlpost.numargs: 5

xmlpost.argname1: pagename
xmlpost.argvalue1: OpenMarket/Xcelerate/Actions/RemoteContentPost

# Notice that you do not need to provide
# the name of the asset type because that information
# is required in the source files for flex assets.

xmlpost.argname2: inifile
xmlpost.argvalue2: futuretense.ini
xmlpost.argname3: authusername
xmlpost.argvalue3: editor
xmlpost.argname4: authpassword
xmlpost.argvalue4: xceleditor
xmlpost.argname5: startmenu
xmlpost.argvalue5: New Product

xmlpost.success: Success
xmlpost.failure: Error
xmlpost.logfile: productdatalog.txt

xmlpost.postdeletefile: y

postpublication: y

post_ASSET_: y
post_ITEMNAME_: y
post_DEFINITION_: y
posttemplate: y

postsku: y
postproductdesc: y
postcaseqty: y
postbulbshape: y
postbulbsize: y
postbasetype: y
postcolortemp: y
postmeanlength: y
postlightcenterlength: y
postreducedwattage: y
postbeamspread: y
postfixturetype: y
```

```
postballasttype: y
postcolorrenderingindex: y
postminstarttemp: y
postpowerfactor: y
posttotalharmonicdist: y
postspreadbeam10h: y
postspreadbeam10v: y
postspreadbeam50h: y
postspreadbeam50v: y
posthalogen: y
postoperatingposition: y
postfilamenttype: y
postbulbimage: y
postbaseimage: y
postfilamentimage: y
postfootnotes: y
postcat1: y
postcat2: y
postprice: y
postvoltage: y
postwattage: y
postlife: y

postParentList: y
```

43.4.6 Sample Flex Asset Source File for RemoteContentPost

This following source file works with the example configuration file immediately preceding this section. This source file creates a lightbulb product named `10004` from the product definition named `Lighting`:

```
<document>

# the first three tags are required
<_ASSET_>Products</_ASSET_>
<_ITEMNAME_>10004</_ITEMNAME_>
<_DEFINITION_>Lighting</_DEFINITION_>

# This tag is required because the publication is
# not set in the configuration file
<publication>AA Illumination</publication>

# This tag assigns a Template asset to the product
<template>Lighting_Detail</template>

# The rest of these tags set flex attribute values for the product
<price>5</price>
<sku>10004</sku>
<productdesc>F4T5/CW</productdesc>
<caseqty>24</caseqty>
<bulbshape>T</bulbshape>
<bulbsize>5</bulbsize>
<basetype>Miniature Bipin (G5)</basetype>
<colortemp>4100</colortemp>
<colorrenderingindex>60</colorrenderingindex>
<bulbimage>BLB-260.gif</bulbimage>
<baseimage>BLB-250.gif</baseimage>
<filamentimage></filamentimage>
<life>6000</life>
<voltage></voltage>
<wattage>4</wattage>
```

```
<cat1>Fluorescent</cat1>
<cat2>Preheat Lamps</cat2>

# this tag sets the immediate parents only
<ParentList>Preheat Lamps</ParentList>

</document>
```

The preceding source file sets several attribute values for the product and sets its immediate parent to Preheat Lamps. This parent must exist.

43.5 Editing Flex Assets with XMLPost

With XMLPost, you can edit the value of an attribute and the asset's parents (either the flex asset's parents or the parent's parents).

You cannot edit attribute assets, flex definition assets, or flex parent definition assets with XMLPost.

To edit the attribute value for a flex asset, the source file needs to include only the name of the asset and the attribute that you want to change.

To edit the attribute value for a flex parent, you must provide the context of a flex asset. The source file must name the flex asset and can then reference just parent and the parent attribute that you want to change. But you must specify a flex asset for XMLPost to start with so that it can work its way through the family tree.

See these topics:

- Configuration Files for Editing Flex Assets

- Source Files for Editing Flex Assets

43.5.1 Configuration Files for Editing Flex Assets

There are two differences in the configuration file for editing a flex asset: the `pagename` argument and an additional tag and property.

Pagename Argument

The `pagename` argument must be set to: `OpenMarket/Gator/XMLPost/modifyData`.

For example:

```
xmlpost.argname1: pagename
xmlpost.argvalue1: OpenMarket/Gator/XMLPost/modifyData
```

You invoke XMLPost from the command line as usual, identifying the configuration file and the source files.

Additional Tag/Property

You can use the following optional tag and property when you are editing a flex asset:

- tag: `<_REMOVE_parentDefinitionName>`

- property: `post_REMOVE_parentDefinitionName`

It removes a parent from the flex asset.

You can use the following optional tag and property to identify the asset to be modified when there are other assets with the same name (_ITEMNAME_), the same asset type (_ASSET_), and the same subtype (_TYPE_) in the same site:

- tag: `<_ID_>`

- property:`post_ID_`

43.5.2 Source Files for Editing Flex Assets

The source file for an edited flex asset does not have to include all the information for that asset, you only have to provide the information that you want to change. Any attributes that you do not specify are not modified in any way.

This section includes the following topics:

- Changing the Value of an Attribute

- Removing an Attribute Value

- Editing Parent Relationships

43.5.2.1 Changing the Value of an Attribute

To change the value of an attribute, you specify the new attribute value in the source file. When XMLPost runs the import, it writes over the old value with the value provided in the source file.

The following sample source file changes two attribute values (bulbshape and bulbsize) for the product named 10004 that was defined in Sample Flex Asset Source File for addData:

```
<document>
<!-- predefined xml tags (required) -->

<_ASSET_>Products</_ASSET_>
<_ITEMNAME_>10004</_ITEMNAME_>
<_TYPE_>Lighting</_TYPE_>


<!-- attribute xml tags -->

<bulbshape>E</bulbshape>
<bulbsize>9</bulbsize>

</document>
```

43.5.2.2 Removing an Attribute Value

- To remove an attribute value and leave it blank, code a line that names the attribute and specify _EMPTY_ as the attribute's value.

  For example:

  ```
  <bulbsize>_EMPTY_</bulbsize>
  ```

  > **Note:** To empty a multivalued attribute, in place of `_EMPTY_` , set the multivalue delimiter which, by default, is the `;` char. For example:
  > `<someattribute>;</someattribute>`.

You can also edit attribute values for parents. Let's say that the bulbsize attribute is set at the parent level. If that were the case, the following lines of code would set two parents and provide a value for bulbsize for each:

```
<_GROUP_SubCategory>All-Weather Lamps;Appliance Lamps
</GROUP_SubCategory>
<bulbsize>All-Weather Lamps=10:Appliance Lamps=8</bulbsize>
```

Option 1

This line of code clears the bulbsize for the All-Weather Lamps parent:

```
<bulbsize>All-Weather Lamps=_EMPTY_:Appliance Lamps=8</bulbsize>
```

Option 2

Alternatively, you could just use this line of code, without repeating the value for Appliance Lamps:

```
<bulbsize>All-Weather Lamps=_EMPTY_</bulbsize>
```

43.5.2.3 Editing Parent Relationships

You can use XMLPost to make the following edits to the parent relationships for a flex asset:

- Add another parent to the existing parents.

- Change a parent from one parent to another.

The `GROUP_parentDefinitionName` tag works differently than the attribute tags:

- When you use an attribute tag, XMLPost writes the new value over the old value.

- When you use a `GROUP_parentDefinitionName` tag, XMLPost does not overwrite an old parent with a new parent, even when the parent definition name is the same. It adds the new parent to the list of parents that the asset has, which may not be what you want.

1. To add another parent to the list of existing parents, include the line of code in the source file. For example:

   ```
   <_GROUP_SubCategory>Blacklights</_GROUP_SubCategory>
   ```

2. To remove a parent, use the `<_REMOVE _>` tag. Note that you must be careful not to remove a required parent unless you are replacing it. For example:

   ```
   <_REMOVE_Processor>Appliance Lamps</_REMOVE_Processor>
   ```

43.6 Deleting Assets with XMLPost

You can delete any asset of any type using XMLPost. Read this to know how:

- Your configuration file must instruct XMLPost to call the `deleteData` element.

  For example:

  ```
  xmlpost.argname1: pagename
  xmlpost.argvalue1: OpenMarket/Gator/XMLPost/deleteData
  ```

- You also need these source file tags and configuration file properties:

<_ASSET_>/post_ASSET_ which identifies the asset type of the asset you want to delete.

<_ITEMNAME>/post_ITEMNAME_ which identifies the asset you want to delete.

When XMLPost uses this posting element, it changes the value in the Status column for that asset to VO for void. (It does not physically remove it from the database).

See these topics:

- Configuration Files for Deleting Assets

- Source Files for Deleting Assets

43.6.1 Configuration Files for Deleting Assets

Here is an example configuration file:

```
xmlpost.xmlfilenamefilter: .xml

xmlpost.url: http://izod19/servlet/ContentServer
xmlpost.numargs: 4
xmlpost.argname1: pagename
xmlpost.argvalue1: OpenMarket/Gator/XMLPost/deleteData
xmlpost.argname2: authusername
xmlpost.argvalue2: user_editor
xmlpost.argname3: authpassword
xmlpost.argvalue3: user
xmlpost.argname4: inifile
xmlpost.argvalue4: futuretense.ini

xmlpost.success: Success
xmlpost.failure: Error
xmlpost.logfile: productdefpostlog.txt
xmlpost.deletefile: y

postpublication: y
post_ASSET_: y
post_ITEMNAME_: y
```

You invoke XMLPost from the command line as usual.

43.6.2 Source Files for Deleting Assets

The source files for deleting assets are short and simple. For example:

```
<document>

<_ASSET_>Products</_ASSET_>
<_ITEMNAME_>Pentium 90</_ITEMNAME_>
<publication>my publication</publication>

</document>
```

This code instructs XMLPost to delete a product asset named Pentium 90 (it changes the status of Pentium 90 to VO, for void).

44

Importing Flex Assets with the BulkLoader Utility

You need the BulkLoader utility to import flex assets when you're setting up your WebCenter Sites system.

Topics:

- About the BulkLoader Utility

- Importing Flex Assets Using a Custom Extraction Mechanism

- Approving Flex Assets with the BulkApprover Utility

44.1 About the BulkLoader Utility

With the BulkLoader utility, you can quickly extract large amounts of flex asset data in a user-defined way from your own data sources. This utility can import that data into the WebCenter Sites database on any of your systems (development, management, testing, or delivery).

The extraction mechanism is abstracted away using a Java interface that customers can implement. BulkLoader invokes methods on this interface to extract input data from your data sources. For backward functional and data compatibility, WebCenter Sites also includes an implementation of this Java interface so that BulkLoader will still be able to extract data from an external JDBC-compliant data source.

The following topics provide information about the BulkLoader utility:

- Understanding BulkLoader Features

- How BulkLoader Works

- About Using the BulkLoader Utility

- Importing Flex Assets from Flat Tables

- When to Use XMLPost to Import Structural Assets

- Creating the Input Table (Data Source)

- Creating the Mapping Table

- Creating the BulkLoader Configuration File

- Running the BulkLoader Utility

- Enabling Access to Imported Assets in the Contributor Interface

- Reviewing Feedback Information

- [Approving and Publishing the Assets to the Delivery System](#)

44.1.1 Understanding BulkLoader Features

Features in BulkLoader include the following:

- Support for a user-defined extraction mechanism, using a Java API. Users can provide a custom implementation of this extraction interface or use the built-in support for extracting from a JDBC data source.

- Support for inserts, voids, and updates of flex asset and group data.

- Support for incremental inserts, voids and updates.

- Performance improvements for higher throughput, using concurrent multi-threaded import operations while data extraction is in progress.

- Support for chunk (slice) processing of input data.

- Support for importing asset data that belongs to multiple flex families.

- Backward functional and data compatibility. Supports importing asset data from an external JDBC source.

44.1.2 How BulkLoader Works

The BulkLoader has been redesigned for higher performance, throughput, and scalability. Instead of reading all input data and then generating output SQL files, BulkLoader reads input data in chunks. As soon as each chunk is read, it is handed over to an import thread while the main BulkLoader thread goes back to read the next chunk. The import thread uses a direct JDBC connection to the WebCenter Sites database. In this way, reading and importing are done in parallel, thereby achieving higher throughput. For scalability, the number of BulkLoader import threads can be increased using the database computer's hardware's additional CPUs and an I/O configuration that supports higher concurrency.

The BulkLoader utility requires a configuration file containing parameters that specify the number of processing threads, the name of the Java class that implements the data extraction interface, commit frequency, the starting unique ID to be used as the asset ID, and more.

The following figures show a client-specific implementation and the built-in ready-to-use implementation supplied by WebCenter Sites.

Figure 44-1   Client-specific implementation of BulkLoader
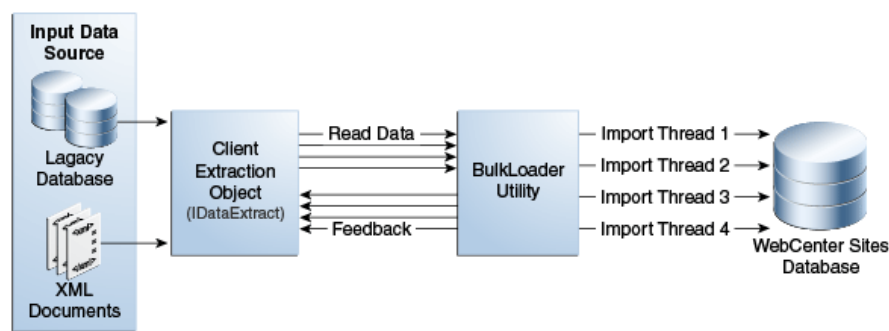
Figure 44-2   Built-in OOTB (out-of-the-box) implementation of BulkLoader
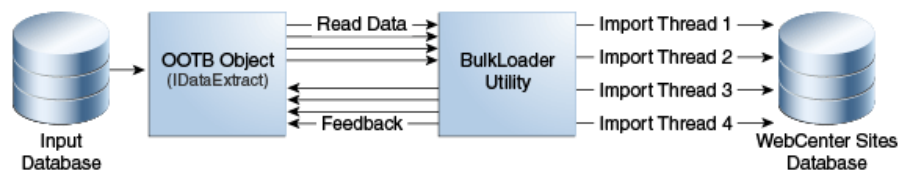
44.1.3 About Using the BulkLoader Utility

There are two ways to use the BulkLoader depending on how you supply input data to import into the WebCenter Sites database.

- To use BulkLoader to import input data from an external JDBC data source, you provide input data in a flat table or view.

- To provide your own way of supplying input data to the BulkLoader, you use a Java object that implements the extraction interface, IDataExtract.

> **Note:**
>
> For reference, sample BulkLoader code is provided on the WebCenter Sites installation medium, in the `Samples` folder. The same folder contains the `readme.txt` file that describes the sample files.

44.1.4 Importing Flex Assets from Flat Tables

This section describes the general procedure that you use to import flex assets with BulkLoader, followed by subsequent sections that describe each step in detail. Using this model, import new flex assets and parents and void assets that were previously imported. This model also supports changing and deleting attribute values for existing assets.

44.1.4.1 The Basic Steps

To import flex assets with the BulkLoader utility:

1. Use XMLPost to import the structural assets into the WebCenter Sites database on the management system. The structural flex assets are as follows: attribute editors, flex attributes, flex parent definitions, flex definitions, and flex parent assets.

2. Write a view or a stored procedure that gives you a view of the source database that you want to import into the WebCenter Sites database as a flat table. This flat table is your source table.

3. In the same source database, create a mapping table with two columns: one column that lists the names of the columns in the source file and the other column that lists the names that are used for those attributes in the WebCenter Sites database.

4. Code a configuration file that identifies the source table and the mapping table.

5. Put the configuration file on a system from which you have access to both the WebCenter Sites database on the management system, and to your source database.

6. Stop the application server on the management system.

7. Run the BulkLoader utility. BulkLoader will import the flex asset data and gives the feedback in a table named `bulk_feedback`, that has been created at the input data source.

8. Restart the application server on the management system.

9. Use the BulkApprover utility to approve all of the assets that were loaded.

> **Note:**
>
> Because the BulkLoader utility is designed for speed, it does **not** check for the existence of the attributes or flex parent definitions or flex definitions. You must import all of the structural asset types before you run the BulkLoader utility.

44.1.4.2 Driver Requirements

The BulkLoader requires JDBC (or Java DataBase Connectivity) drivers, which are not provided by Oracle Corporation. You must obtain JDBC drivers for both the source database and the destination database, that is, the WebCenter Sites database. For an ODBC-compliant source database, use a JDBC-ODBC bridge, which is included as part of the Java SDK.

44.1.4.3 Requirement for DB2

Run the `usejdbc2.bat` file on the client computer before you can use BulkLoader. Run the batch file once, and then run BulkLoader as usual.

44.1.5 When to Use XMLPost to Import Structural Assets

Use XMLPost and the `RemoteContentPost` posting element to import the structural assets into the WebCenter Sites database on the management system. Import assets of the following types:

- attribute editors
- flex attributes
- flex parent definitions
- flex definitions
- flex parent assets

See About Importing the Structural Asset Types in the Flex Model.

44.1.6 Creating the Input Table (Data Source)

You must create input flat tables (data sources) for holding all new asset data and for holding update data. These are flat tables/views in which each row corresponds to a single flex asset item and each column corresponds to a flex attribute asset for the BulkLoader utility.

There is no requirement regarding the names of columns in the data source, but you must supply a separate mapping table, described in Creating the Mapping Table.

This section includes the following topics:

- **Inserts**

- **Updates**

44.1.6.1 Inserts

The name of the data source table is specified by the `inputTable` parameter in the configuration file.

The source table must also include the names of the following four columns, which you specify in the configuration file with the following properties:

- `inputTableTemplateColumn`: The name of the column in the source table that holds the names of the flex definitions.

- `inputTableNameColumn`: The name of the column in the source table that holds the names of the flex assets. The name of this column cannot exceed 64 characters.

- `inputTableDescriptionColumn`: The name of the column in the source table that holds the description of the flex assets.

- `inputTableGroupsColumn`: The name of the column in the source table that holds the parent names. Each value in this column can include multiple flex parent names, separated by the `multivalueDelimeter` character, which is defined in the configuration file.

Note that you can optionally specify the name of the column that serves as a unique identifier for each input item, using the following parameter in the configuration file: `inputTableUniqueIdColumn`. If there is no value assigned for this parameter, BulkLoader will generate a unique identifier for each input item and store it in a mapping table (`bulkloader_ids`) in the WebCenter Sites database.

The following is an example of a source table (input table):

Figure 44-3   Sample Source Input Table

Based on the column names in this source table, the source table properties in the corresponding configuration file would be set as follows:

```
inputTableTemplateColumn=SNSTEMPLATE
inputTableNameColumn=SNSNAME
inputTableDescriptionColumn=SNSDESCRIPTION
inputTableGroupsColumn=SNSGROUPS
```

44.1.6.2 Updates

To update attribute data for existing assets, to add new parents or delete existing parents for existing assets, you use the update parameter.

Use the `inputTableForUpdates` parameter in the configuration file to specify the name of the data source table. The source table must also include the names of the following three columns, which you specify in the configuration file with the following properties:

- `inputTableForUpdatesUniqueIdColumn`: The name of the column in the source table that uniquely identifies the flex asset or parent in the WebCenter Sites database.

- `inputTableForUpdatesDeleteGroupsColumn`: The name of the column in the source table that specifies a list of parents to be deleted for the current flex asset.

- `inputTableForUpdatesAddGroupsColumn`: The name of the column in the source table that specifies a list of parents to be added for the current flex asset.

BulkLoader interprets column values as follows when applying updates to the attributes:

- A null value in a specific attribute column indicates that the attribute for the current flex asset should be deleted. For example, a null value in the `deletegroups` column indicates that no parents have to be deleted. A null value in the `addgroups` column indicates that no parents have to be added.

- A non-null value indicates that the existing attribute value should be replaced with the given value. For example, a non-null value in the `deletegroups` column specifies a list of parents to be deleted. A non-null value for `addgroups` denotes the addition of new parents to a given flex asset.

44.1.7 Creating the Mapping Table

You must also create a mapping table for the BulkLoader utility, and it must have the following two columns:

- A column that holds the names of the flex attribute columns in your flat data source.

- A column that holds their corresponding names in the WebCenter Sites database.

The mapping table provides a one-to-one correspondence between these two columns. For example, your source table might have a column of vendor names with an automatically generated name like `A96714328445` that maps to a product attribute asset named, simply, `VENDOR_ID`.

You include the following configuration file properties for the mapping table:

- `inputAttributeMapTable`: The name of the mapping table file.

- `inputAttributeMapTableKeyCol`: The name of the column in the mapping table that lists the attribute names in the source table.

- `inputAttributeMapTableValCol`: The name of the column that lists the corresponding attribute asset names in the WebCenter Sites database.

The following is an example of a mapping table:

Figure 44-4   Sample Mapping Table

Based on the column names in this source table, the source table properties in the corresponding configuration file would be set as follows:

```
inputAttributeMapTable=93ATTR_MAP
inputAttributeMapTableKeyCol=SOURCENAME
inputAttributeMapTableValCol=ATTRIBUTENAME
```

44.1.8 Creating the BulkLoader Configuration File

You configure the BulkLoader utility by creating a configuration file for it that has the properties described in this section. You can name the file anything you want.

You set the properties in the file according to the following syntax:

```
property=value
```

> **Note:**
>
> All property names and values in the configuration file are case-sensitive.

This section includes the following topics:

- BulkLoader Configuration File Properties
- Setting the initID Parameter
- Example Configuration File

44.1.8.1 BulkLoader Configuration File Properties

This table describes properties in a BulkLoader configuration file:

Table 44-1   BulkLoader Configuration File Properties

Property Name	Required/ Optional	Comments
`maxThreads`	Required	The maximum number of concurrent processing threads. This can be the number of database connections to the WebCenter Sites database server. Use as many threads as the number of CPUs on the database host. For a single CPU database host, set it to 2. Example: 4
`dataSliceSize`	Required	Number of items retrieved in one read request; this number will also be processed by a single processing thread. Example: `2000`
`dataExtractionImplClass`	Required	User-specific Implementation class for data extraction API. Needs a constructor with (`String configFilename`) signature. The one mentioned here is a reference implementation class for backward compatibility. Data in flat tables. Default value (ready-to-use): `com.openmarket.gatorbulk.objects.DataExtractImpl`
`initId`	Required	Starting WebCenter Sites ID used the very first time BulkLoader operates; subsequently will use the value from `idSyncFile`. Example: `800000000000`
`idSyncFile`	Required	Next available WebCenter Sites ID is saved in this file; updated during a BulkLoader session. Example: `C:\FutureTense\BulkLoaderId.txt`
`idPoolSize`	Required	Each time BulkLoader needs to generate WebCenter Sites IDs, it collects this many IDs and caches in memory. A good estimate is (number of assets * average number of attributes *2). Example: `1000`
`commitFrequency`	Required	Number of flex asset groups to be part of a database transaction. Example: `100`
`outputJdbcDriver`	Required	The name of the JDBC driver class to access the WebCenter Sites database. The value here reflects the Oracle 9.0 driver. Example: `oracle.jdbc.driver.OracleDriver`

Table 44-1   (Cont.) BulkLoader Configuration File Properties

Property Name	Required/ Optional	Comments
outputJdbcURL	Required	The JDBC URL. The following example value is a typical type2 oracle JDBC driver URL: `Jdbc:oracle:oci8:@foo`
outputJdbcUsername	Required	WebCenter Sites database user name.
outputJdbcPassword	Required	WebCenter Sites database user password.
inputTable	Required	Name of the flat, input table from which new asset data is inserted.
inputAttributeMapTable	Required	Name of the mapping table that lists the source table columns and the corresponding attribute names.
inputAttributeMapTableKeyCol	Required	The name of the column in the mapping table that lists the source table column names. For example: `inputAttributeMapTableKeyCol=SOURCE NAME`
inputAttributeMapTableValCol	Required	The name of the column in the mapping table that lists the corresponding attribute names. For example: `inputAttributeMapTableValCol=ATTRIB UTENAME`
inputTableDescriptionColumn	Required	The name of the column in the source table that contains the descriptions of the flex assets. For example: `inputTableDescriptionColumn=SNSDESC RIPTION`
inputTableGroupsColumn	Required	Name of the column in the source table that contains the names of parents. Each value can include several parents, separated by the `multivalueDelimeter` character, which is defined in the configuration file. For example: `inputTableGroupsColumn=SNSGROUP`
inputTableNameColumn	Required	The name of the column in the source table that contains the name of the product (or advanced article or advanced image) for each row. For example: `inputTableNameColumn=SNSNAME`

Table 44-1   (Cont.) BulkLoader Configuration File Properties

Property Name	Required/ Optional	Comments
inputTableTemplateColumn	Required	The name of the column in the source table that contains the flex definitions. For example: `inputTableTemplateColumn=SNSTEMPLATE`
createdby	Required	The user name that you want to be entered in the `createdby` field for your flex assets. For example: `createdby=editor`
multivalueDelimeter	Required	The delimiter that separates multiple attribute values. The default character is the semicolon (*;*). For example: `multivalueDelimiter=;`
siteName	Required	The name of the site. All products will behave as if they were created under this site. For example: `siteName=AA Illumination`
status	Required	The status code for all imported flex assets. You should set this to PL for imported. For example: `status=PL`
tableProducts	Required	Name of the flex asset type as defined in the WebCenter Sites database.
inputTableUniqueIdColumn	Required	Name of the column in the source table that serves as a unique identifier when importing a new flex asset. This will be used for any subsequent updates and void operations. To allow BulkLoader to generate unique identifiers, leave this value blank.
targetName	Required	Name of the publish target, as defined in WebCenter Sites.
renderTemplate	Optional	Name of the template used for rendering flex assets (deprecated).
inputFeedbackTable	Required	Name of the table that BulkLoader creates and uses for recording the processing feedback for every input item that was processed. Note that this table is created in the input data source.

Table 44-1   (Cont.) BulkLoader Configuration File Properties

Property Name	Required/ Optional	Comments
`inputTableForUpdates`	Optional	Needed only if an update action is specified when running the BulkLoader utility. Otherwise, this can be an empty value. This is the name of the source table that contains attributes and parents that need updates.
`inputTableForUpdatesUniqueIdColumn`	Optional	Needed only if an update action is specified when running the BulkLoader utility. This is the name of the column in the source table that specifies a unique identifier for the flex asset.
`inputTableForUpdatesDeleteGroupsColumn`	Optional	Needed only if update action is specified and you have one or more flex assets that need one or more parents to be deleted. This is the name of the column in the source table that specifies the list of parents to be deleted.
`inputTableForUpdatesAddGroupsColumn`	Optional	Needed only if an update action is specified and you have one or more flex assets that need one or more parents to be added. This is the name of the column in the source table that specifies a list of parents to be added.
`inputLimitRows`	Optional	Needed only for testing. Limits the number of input items processed for each action (insert, void, or update).
`updatedby`	Optional	The user name that you want to be entered in the `createdby` field for your flex assets. For example: `updateby=editor`
`updatedstatus`	Optional	The status code for all updated flex assets. This must be set to `ED`. For example: `updatestatus=ED`

44.1.8.2 Setting the initID Parameter

The `initID` parameter is the seed value that the BulkLoader starts at and increments from when creating a unique asset ID for each asset. You must choose a seed value number that allows the BulkLoader to create a contiguous block of ID numbers that cannot cause ID conflicts with existing (or future) asset ID numbers that are generated by WebCenter Sites.

Currently, WebCenter Sites starts at 1 trillion for the asset IDs that it creates. To be sure that you won't have conflicts, select a number low enough that when the BulkLoader utility is done, the highest ID number is under 900,000,000,000.

The BulkLoader creates one asset for each row/column value in the data source table. Each output table row requires its own unique asset ID.

Use these guidelines to determine the approximate number of asset IDs that are created by the BulkLoader utility:

- Five rows for each flex asset, plus

- Two rows per attribute for each flex asset

For example, if your data source table contains the following:

- 10,000 product assets

- 20 attributes per product (as determined by the product definition)

- 10 inherited attributes per product (as determined by the product parent definitions)

Then you have to allow for the following number of IDs:

(5 x 10,000) + (2 x 30 x 10,000) = 50,000 + 600,000 = 650,000 asset IDs

If your initID value is 800,000,000,000, then the BulkLoader creates ID numbers ranging from 800,000,000,000 to approximately 800,000,650,000.

44.1.8.3 Example Configuration File

The following is an example of a BulkLoader configuration file:

```
# New BulkLoader configuration for backward compatibility
#
# input datasource configuration
inputJdbcDriver=sun.jdbc.odbc.JdbcOdbcDriver
inputJdbcURL=jdbc:odbc:access-db-conn
inputJdbcUsername=
inputJdbcPassword=
#
# Source tables
#
inputTable=PRD_FLAT_50000
inputAttributeMapTable=PRD_FLAT_ATTRIBUTE_MAP
inputAttributeMapTableKeyCol=SOURCENAME
inputAttributeMapTableValCol=ATTRIBUTENAME
#
# input column names
#
inputTableTemplateColumn=CCTemplate
inputTableNameColumn=CCName
inputTableDescriptionColumn=CCDescription
inputTableGroupsColumn=CCGroups
#
# WebCenter
Sites database
#
#  This database is always used for looking up Attributes, #  Product Types and
Product Group Types. #  Data is imported into this database.
#
outputJdbcDriver=oracle.jdbc.driver.OracleDriver
outputJdbcURL=jdbc:oracle:oci8:@foo
outputJdbcUsername=csuser
outputJdbcPassword=csuser
#
# Data-specific settings
#
siteName=AA Illumination
targetName=Mirror Publish to burst37
initId=800000000000
createdby=user_designer
```

```
status=PL
renderTemplate=CLighting Detail
MAX_ATTRIBUTES=100
multivalueDelimiter=;
commitFrequency=50
#
# The following denotes the flex asset type that we are importing.
tableProducts=Products
#
# Additional information needed for BulkLoader
maxThreads=2
# dataSliceSize 0 means read all input data in one slice.
dataSliceSize=500
dataExtractionImplClass=com.openmarket.gatorbulk.objects.DataExtractImpl
idSyncFile=C:\\FutureTense50\\bulk_uniqueid.dat
idPoolSize=50000
# For inserts
inputTableUniqueIdColumn=
inputFeedbackTable=bulk_feedback
# For updates
inputTableForUpdates=prod_flat_2_upd
inputTableForUpdatesUniqueIdColumn=input_id
inputTableForUpdatesDeleteGroupsColumn=CCGroups
inputTableForUpdatesAddGroupsColumn=
inputLimitRows=1000
############################################################
```

44.1.9 Running the BulkLoader Utility

Before you begin, be sure that you have the appropriate JDBC drivers for **both** your **source** database and your **target** WebCenter Sites database.

To run the BulkLoader utility:

1. In the Admin interface, disable Lucene search engine indexing. See Disabling the Lucene Search Engine in *Administering Oracle WebCenter Sites*.

2. Put the configuration file on a system from which you have access to both the WebCenter Sites database on the management system, and to your source database.

3. Stop the application server on the management system.

4. To get Bulkloader up and running, set the following directories to classpath:
    `<ORACLE_HOME>\wcsites\webcentersites\sites-home\lib\*` and
    `<ORACLE_HOME>\oracle_common\modules\clients\*`.

    If WebCenter Sites is using the SQL Server database, include in classpath:
    `Oracle_Home\oracle_common\modules\*`.

5. Enter the following command, all on a single line, with paths that are appropriate for your installation:

    For UNIX

    ```
    java -ms16m -mx256m -cp <ORACLE_HOME>/wcsites/webcentersites/sites-home/lib/
    *:<ORACLE_HOME>/oracle_common/modules/clients/*
    com.openmarket.gatorbulk.objects.BulkLoader config=bulkloader.ini action=<insert|
    void|update> validate=<yes|no>
    ```

    For Windows

```
java -ms16m -mx256m -cp <ORACLE_HOME>\wcsites\webcentersites\sites-home\lib
\*:<ORACLE_HOME>\oracle_common\modules\clients\*
com.openmarket.gatorbulk.objects.BulkLoader config=bulkloader.ini action=<insert|
void|update> validate=<yes|no>
```

Note that the action parameter specifies what BulkLoader needs to do: `insert`, `void`, or `update`. Setting the validate parameter to `yes` makes BulkLoader do extra validations during updates and voids. You may also have to increase the memory for the JVM, depending on the size of your input data.

6. Examine the screen output to be sure that the BulkLoader utility was able to connect to the appropriate database.

44.1.10 Enabling Access to Imported Assets in the Contributor Interface

To access assets imported with the BulkLoader utility from the Oracle WebCenter Sites: Contributor interface, complete the following steps:

1. Re-enable the Lucene search engine indexing. See Setting Up Search Indices in *Administering Oracle WebCenter Sites*.

2. Add the assets you imported with the BulkLoader utility to the search index. See Adding Asset Types to the Search Index in *Administering Oracle WebCenter Sites*.

3. Verify the assets are accessible from the Contributor interface, by conducting a search, in the Contributor interface, for the assets you imported with the BulkLoader utility. For instructions about conducting searches in the Contributor interface, see Finding and Organizing Assets in *Using Oracle WebCenter Sites*.

44.1.11 Reviewing Feedback Information

After the BulkLoader utility completes an operation, review the feedback information in the `bulk_feedback` table that is located in your input data source. That table contains information about all the input items that BulkLoader processed.

After reviewing that information, take any corrective actions that might be necessary. If you modify any of your input data, run BulkLoader again to verify that the errors were corrected.

44.1.12 Approving and Publishing the Assets to the Delivery System

Use the BulkApprover utility to approve the assets that you just loaded. See Using BulkApprover.

44.2 Importing Flex Assets Using a Custom Extraction Mechanism

Sometimes users need alternative mechanisms to provide input asset data to BulkLoader. In such cases, the data may have to be gathered from multiple types of sources, such as XML documents, files, and legacy databases. To accomplish that, users can implement their own mechanism to provide data to BulkLoader, using the Java interface `com.openmarket.bulkloader.interfaces.IDataExtract`, which is provided with WebCenter Sites.

A user can implement a Java object supporting `IDataExtract` and specify the Java object in the BulkLoader configuration file. BulkLoader will then invoke methods on this interface to initialize a read request, to repetitively read chunks of input data and then signal the end of the read request. This interface also has a method that provides import feedback from the BulkLoader utility, which can be used by the input provider to know the status of import and know any errors that may occur during import.

There are three Java interfaces that can help users with custom implementations of `IDataExtract`:

- `IDataExtract`: Required for any custom extraction.

- `IPopulateDataSlice`: Provides data to the BulkLoader utility. A container object supporting this interface is created by BulkLoader and passed into the client.

- `IFeedback`: Provides the status of each input item that has been processed by the BulkLoader. A feedback object that is created and populated by BulkLoader import thread is passed into the client.

See these topics:

- IDataExtract Interface

- IPopulateDataSlice

- IFeedback Interface

44.2.1 IDataExtract Interface

This interface is required for any custom extraction.

The following is a sample code that implements this interface:

```
com.openmarket.gatorbulk.interfaces.IDataExtract

package com.openmarket.gatorbulk.interfaces;
import java.util.Iterator;

/**
 * To be implemented by input data provider.
 * Interface for extracting data from an input source
 * for BulkLoader.
 * BulkLoader loads an object supporting this interface and invokes
 * the GetNextInputDataSet() method on this interface repeatedly to
 * fetch data in batches.
 */

public interface IDataExtract {

  public  final int HAS_DATA   = 100;
  public  final int NO_DATA = 101;

  public  final int SUCCESS = 0;
  public  final int ERROR = -1;

  public  final int INSERT_ASSETS = 1000;
  public  final int VOID_ASSETS = 1010;
  public  final int UPDATE_ASSETS = 1020;
  public  final int NONE_ASSETS = 1030;

  /**
   * Begin requesting input data; tells the client to
   * start the database query, get a cursor, etc.
   * @param requestType
   *     IDataExtract.INSERT_ASSETS,
   *     IDataExtract.VOID_ASSETS,
   *     IDataExtract.UPDATE_ASSETS
```

```
 * @param sliceOrNot  true/false
 * true - if data will be requested in batches
 * false - data will be requested all in one attempt
 * @param sliceSize  >0 number of rows to be
 * retrieved in one data set
 * @return none
 * @exception java.lang.Exception
 */

public void InitRequestInputData(int requestType,
boolean sliceOrNot, int sliceSize) throws Exception ;

/**
 * Get a set/slice of input data records.
 * @param dataSlice object to be populated using the
 * methods from IPopulateDataSlice
 * @return IDataExtract.HAS_DATA when dataSlice has some data,
 *     IDataExtract.NO_DATA when there is no data,
 *     IDataExtract.ERROR when there is an error
 * @exception java.lang.Exception
 */

public int  GetNextInputDataSet(IPopulateDataSlice dataSlice)
throws Exception;

/**
 * Signal the end of extracting data for given request type
 * @param requestType
 *     IDataExtract.INSERT_ASSETS,
 *     IDataExtract.VOID_ASSETS,
 *     IDataExtract.UPDATE_ASSETS
 * @return none
 * @exception java.lang.Exception
 */

public void EndRequestInputData(int requestType)
  throws Exception;

/**
 * Update the client as to what happened to input data
 * processing. Note that this method would be called by multiple
 * threads, with each thread passing its own IFeedback
 * handle. The implementor of this method should write
 * thread-safe code.
 * @param requestType
 *     IDataExtract.InsertAsset,
 *     IDataExtract.VoidAsset,
 *     IDataExtract.UpdateAsset
 * @param processingStatus - An object containing processing
 * status for all items in one dataset. The implementor of this
 * interface should invoke the IFeedback interface
 * methods on processingStatus to get status for individual
 * rows. This method will be invoked by multiple BulkLoader
 * threads, so make sure this method is implemented in a
 * thread-safe way.
 * @return none
 * @exception java.lang.Exception
 */

void UpdateStatus(int requestType, IFeedback  processingStatus)
```

```
    throws Exception;
}
```

Implementation Notes for IDataExtract

The Java object implementing `IDataExtract` needs to have a constructor with a string parameter. BulkLoader will pass the name of its configuration file to the constructor when instantiating this object.

The method `UpdateStatus(..)` is invoked by multiple BulkLoader threads, so the implementation of this method should be thread-safe.

This table lists and describes the configuration parameters for the BulkLoader utility when using custom data extraction method:

Table 44-2   Configuration Parameters for BulkLoader

Property Name	Required/ Optional	Comments
`maxThreads`	Required	The maximum number of concurrent processing threads. This can be the number of database connections to the WebCenter Sites database server. Use as many threads as the number of CPUs on the database host. For a single CPU database host, set it to 2. Example: 4
`dataSliceSize`	Required	Number of items retrieved in one read request; this number will also be processed by a single processing thread. Example: `2000`
`dataExtractionImplClass`	Required	User-specific Implementation class for data extraction API. Needs a constructor with (String configFilename) signature. The one mentioned here is a reference implementation class for backward compatibility. Data in flat tables. Default value (ready-to-use): `com.openmarket.gatorbulk.objects.DataExtractImpl`
`initId`	Required	Starting WebCenter Sites ID used the very first time BulkLoader operates; subsequently will use the value from `idSyncFile`. Example: `800000000000`
`idSyncFile`	Required	Next available WebCenter Sites ID is saved in this file; updated during a BulkLoader session. Example: `C:\FutureTense\BulkLoaderId.txt`
`idPoolSize`	Required	Each time BulkLoader needs to generate WebCenter Sites IDs, it collects this many IDs and caches in memory. A good estimate is (number of assets * average number of attributes *2). Example: `1000`

Table 44-2   (Cont.) Configuration Parameters for BulkLoader

Property Name	Required/ Optional	Comments
commitFrequency	Required	Required. Specifies when "COMMIT" statements will be inserted into the generated SQL file. A value of 0 means that "COMMIT" statements will be inserted every 50 lines (the default); any positive integer specifies the number of lines between each "COMMIT" statement. For example: commitFrequency=5 (A COMMIT statement will be inserted for every 5 lines of SQL code.)
outputJdbcDriver	Required	The name of the JDBC driver class to access the WebCenter Sites database. The value here reflects the Oracle 9.0 driver. Example: oracle.jdbc.driver.OracleDriver
outputJdbcURL	Required	The JDBC URL. The following example value is a typical type2 oracle JDBC driver URL: Jdbc:oracle:oci8:@foo
outputJdbcUserna me	Required	WebCenter Sites database user name.
outputJdbcPasswo rd	Required	WebCenter Sites database user password.

44.2.2 IPopulateDataSlice

The following is the sample code that implements this interface:

```
com.openmarket.gatorbulk.interfaces.IPopulateDataSlice

package com.openmarket.gatorbulk.interfaces;

import java.sql.Timestamp;

/**
 * To be implemented by Oracle Corporation
 * Interface to populate a dataSlice by the client.
 * BulkLoader creates an object implementing this interface and then
 * hands it over to the client, which uses this interface's methods
 * to populate that object with input data records.
 */

public interface IPopulateDataSlice {

/**
 * Creates a new input data object to hold all the data for a
 * flex asset and makes it the current object. This method is
 * invoked repetitively to populate this object with flex asset
 * input data. Each invocation is to be followed by Set..()
```

```
 * methods and AddAttribute..() methods to supply data for one
 * flex asset.
 */

public void AddNewRow();

/**
 * Specify a unique identifier for flex asset input data
 * @param id   user-specific unique identifier
 * @exception java.lang. Exception thrown if any unique-id
 * validation is enabled.
 */

public voidSetAssetUniqueId(String id);

/**
 * Specify the name of the site with which the current flex
 * asset is created or to be created under.
 * @param sitename   name of the site
 */

public void SetSiteName(String sitename);

/**
 * Set the asset type for the flex asset.
 * @param flexAssetType asset type as defined in WebCenter
Sites system
 */

public void SetFlexAssetType(String flexAssetType);

/**
 * Specify the name of the parent for the current flex asset.
 * Use this method repeatedly to add a list of parent names.
 * @param groupName   name of a parent that the current asset
 * inherits some of its attributes from.
 */

public void AddParentGroup(String groupName);

/**
 * Specify the name of the parent to be deleted for the current
 * flex asset.
 * Use this method repeatedly to add a list of parent names.
 * @param groupName - name of a parent that the current asset
 * inherited some of its attributes from.
 */

public void AddParentGroupForDelete(String groupName);

/**
 * Specify definition asset name for the current flex asset.
 * @param definitionAssetName name of the flex definition asset
 */

public void SetDefinitionAssetName(String definitionAssetName);

/**
 * Specify name of the flex asset.
 * @param name - name of the flex asset.Should be unique in
 * a flex asset family
```

```
 */

public void SetAssetName(String name);

/**
 * Specify description for the flex asset
 * @param description   description
 */

public void SetAssetDescripiton(String description);

/**
 * Specify WebCenter
Sites username with which this flex asset is being
 * processed
 * @param username WebCenter
Sites username
 */

public void SetCreatedByUserName(String userName);

/**
 * Set WebCenter
Sites status code for this asset
 * @param status
 */

public void SetAssetStatus(String status);

/**
 * Set template name
 * @param template  WebCenter
Sites template name
 */

public void SetRenderTemplateName(String template);

/**
 * Specify startMenu for workflow participation
 * @param startMenuName  start menu name for this flex asset
  */

public void SetStartMenuName(String startMenuName);

/**
 * WebCenter
Sites
 * Specify publish approval target name
 * @param targetName   approval target name
 */

public void SetApprovalTargetName(String targetName);

/**
 * Add a name/value pair to specify a WebCenter
Sites attribute
 * of type 'text' for the current input object.
 * Call this method more than once, if this is a
 * multi-valued attribute.
 * @param attrName  attribute name as defined in the WebCenter
Sites
```

```
 * database for the flex asset being processed
 * @param value java.lang.String
 */

public void AddAttributeValueString(String attrName, String value);

/**
 * Add a name/value pair to specify a WebCenter
Sites attribute
 * of type 'date' for the current input object.
 * Call this method more than once, if this is a
 * multi-valued attribute.
 * @param attrName  attribute name as defined in the WebCenter
Sites
 * database for the flex asset being processed
 * @param value  java.sql.Timestamp
 */

public void AddAttributeValueDate(String attrName, Timestamp value);

/**
 * Add a name/value pair to specify an attribute for the current
 * input object.
 * Call this method more than once, if this is a multi-valued *attribute
 * @param attrName  attribute name as defined in WebCenter
Sites database
 * for the flex asset being processed
 * @param value  java.lang.Double
 */

public void AddAttributeValueDouble(String attrName, Double value);

/**
 * Add a name/value pair to specify a WebCenter
Sites attribute
 * of type 'money' for the current input object
 * Call this method more than once if this is a
 * multi-valued attribute
 * @param attrName  attribute name as defined in WebCenter
Sites database
 * for the flex asset being processed
 * @param value  java.lang.Float
 */

public void AddAttributeValueFloat(String attrName, Float value);

/**
 * Add a name/value pair to specify a WebCenter
Sites attribute
 * of type 'int' for the current input object.
 * Call this method more than once, if this is a
 * multi-valued attribute.
 * @param attrName  attribute name as defined in WebCenter
Sites
 * database for the flex asset being processed
 * @param value  java.lang.Integer
 */

public void AddAttributeValueInteger(String attrName,
  Integer value);
```

```
/**
 * Add a name/value pair to specify any WebCenter
Sites attribute for the
 * current input object.
 * Use the datatype-specific methods above instead of this
 * method, as this one is for
 * supporting any other new types in future.
 * Call this method more than once, if this is a
 * multi-valued attribute
 * @param attrName  attribute name as defined in the WebCenter
Sites
 * database for the flex asset being processed.
 * @param value  java.lang.Object
 */

public void AddAttributeValueObject(String attrName,
  Object value);
}
```

44.2.3 IFeedback Interface

The following is the sample code that implements this interface:

```
com.openmarket.gatorbulk.interfaces.IFeedback

package com.openmarket.gatorbulk.interfaces;

import java.util.Iterator;

/**
 * To be implemented by Oracle Corporation
 * Interface for the BulkLoader client to get the status of
 * processing request to insert/void/update flex assets.
 */

public interface IFeedback {
  public final int ERROR=-1;
  public final int SUCCESS=0;
  public final int NOT_PROCESSED=1;

/**
 * Get a list of keys from input data slice that has
 * been processed
 * @return java.util.Iterator
 */

public Iterator GetInputDataKeyValList();

/**
 * Get WebCenter
Sites asset ID for given input identifier
 * @param inputDataKeyVal key value of the unique identifier
 * in the input data record
 * @return Get the associated asset ID from the WebCenter
Sites system.
 * null if missing.
 */

public String GetWebCenter SitesAssetId(String inputDataKeyVal);

/**
```

```
 * Get the processing status for the input data record
 * identified by a key
 * @param inputDataKeyVal key value of the unique identifier
 * column in the input data record
 * @return ERROR - processed but failed, SUCCESS - processed
 * successfully, NOT_PROCESSED - unknown item or not part of
 * the processing dataset.
 */

public int GetStatus(String inputDataKeyVal);

/**
 * Get the associated error message for a given key,
 * unique identifier in input data
 * @param inputDataKeyVal  unique identifier for input data
 * @return error message, if GetStatus() returned ERROR
 * or NOT_PROCESSED
 */

public String GetErrorDescription(String inputDataKeyVal);
```

> **Note:**
>
> When you implement a custom extraction method, you use the same previously described procedures to run BulkLoader. See About Using the BulkLoader Utility.

44.3 Approving Flex Assets with the BulkApprover Utility

BulkApprover quickly and easily approves large numbers of flex assets that you have loaded into the system via BulkLoader.

BulkApprover can perform the following tasks:

- Notify the approval system of all updates and deletions that were made during a previous BulkLoader session.

- Approve all newly loaded flex assets for one or more publishing targets.

- Mark all newly loaded flex assets as published for a given mirror destination, without actually publishing the assets. For example, to bypass a long mirror publishing session, copy selected assets from the content management database to a mirror destination on the delivery system and have BulkApprover mark the assets as published to the mirror destination.

> **Note:**
>
> Only users with the xceladmin role can run BulkApprover.

This section includes the following topics:

- Configuring BulkApprover

- Using BulkApprover

44.3.1 Configuring BulkApprover

Before running BulkApprover for the first time, you must create a configuration file for the utility. You can create a separate `BulkApprover.ini` file on a system that can access WebCenter Sites database, or you can append the BulkApprover configuration information to one of BulkLoader's `.ini` files.

This table lists required and optional configuration parameters:

Table 44-3   BulkApprover Configuration Parameters

Parameter	Description
`bulkApprovalURL` (Required)	The URL on the host server that has the data imported with BulkLoader. The correct value is as follows: `http://<myServer>/cs/ContentServer? pagename=OpenMarket/Xcelerate/Actions/ BulkApproval` where `<myServer>` is the name of the host server.
`adminUserName` (Required)	The WebCenter Sites username of a user with the `xceladmin` role.
`adminUserPassword` (Required)	The password for `adminUserName`.
`approvalTargetList` (Required)	A list of destinations that the assets are to be approved for. For destination names, see the **Publish** option on the **Admin** tab, or the name column of the `pubtarget` table. Separate each destination with the delimiter that you specify in the `multiValueDelimiter` parameter. The syntax is: `name1<multiValueDelimiter>name2<multiValueDelimiter>name3`
`multiValueDelimiter` (Required)	A delimiter that you select. Use this delimiter to separate the approval targets that you specify in the `appovalTargetList` parameter.
`assetIdSqlFilter` (Optional)	A statement that can be appended to a SQL `WHERE` clause to filter asset IDs. For example: `asset_id%20=0` or `asset_id%20!=0`
`debug` (Optional)	Turn BulkApprover debugging on and off. A value of `true` turns debugging on. Leave this parameter blank for no debugging. Debug messages are written to the file specified in the `output_file` parameter of the command line.

Table 44-3   (Cont.) BulkApprover Configuration Parameters

Parameter	Description
assetschunksize (Optional)	Specifies the number of assets that are approved in a single transaction. For example, setting this property to 20 means that assets will be approved in groups of 20. Setting this property helps prevent session timeouts. Default value: 25
outputJdbcDriver (Required)	The name of the JDBC driver class to access the WebCenter Sites database. Example: oracle.jdbc.driver.OracleDriver
outputJdbcURL (Required)	The JDBC URL. The following example value is a typical type 2 oracle JDBC driver URL: Jdbc:oracle:oci8:@foo
outputJdbcUsername (Required)	WebCenter Sites database user name.
outputJdbcPassword (Required)	WebCenter Sites database user password.

Sample BulkApprover.ini File

The following sample shows the proper syntax of the BulkApprover configuration parameters:

```
bulkApprovalURL=http://MyServer/cs/ContentServer?pagename=OpenMarket/Xcelerate/
Actions/BulkApproval
multiValueDelimiter=;;;;;;
assetIdSqlFilter=
assetsChunkSize=3
debug=true
outputJdbcDriver=oracle.jdbc.driver.OracleDriver
outputJdbcURL=jdbc:oracle:thin:@19zln:1521:MyServer
#outputJdbcUsername=izod10
outputJdbcUsername=ftuser3
outputJdbcPassword=ftuser3
```

44.3.2 Using BulkApprover

After you have configured the BulkApprover utility, use it to approve assets that were imported into the database via the BulkLoader utility.

BulkApprover runs from the command line. Parameters are described in the following table and included in the example that follows.

Table 44-4   BulkApprover Parameters

Command-Line Parameter	Description
config	The name of the file that stores your BulkApprover configuration information; for example, BulkApprover.ini.

Table 44-4   (Cont.) BulkApprover Parameters

Command-Line Parameter	Description
`action`	The action or actions that you want BulkApprover to perform. You must set this parameter to `notify` or `approve`. Add other actions, as necessary. To have BulkApprover perform multiple actions, name the actions in a comma-separated list.
	Valid values:
	• `notify`: Notifies the approval system about all updates and voids processed during a previous BulkLoader session.
	• `approve`: Instructs BulkApprover to approve all of the assets that it processes for the given publishing destination(s).
	• `mark_publish`: Marks all of the assets that it processes as published to a given mirror destination, without actually publishing the assets. Specify the publishing targets in the `approvalTargetList` parameter in the BulkApprover configuration file. Do not include this parameter if the assets should not be marked published.
`output_file`	The name of the log file that contains all output from the server; for example, `bulkapprover.txt`.

To run BulkApprover, set paths as shown in the following example.

This example uses Windows syntax. For UNIX-based systems including Linux and Solaris, the forward-slash (/) and colon (:) should be used as separators:

```
java -ms16m -mx64m -cp <ORACLE_HOME>\wcsites\webcentersites\sites-home\lib
\*;<ORACLE_HOME>\oracle_common\modules\clients\*;<path to jtds-1.2.2.jar>
\jtds-1.2.2.jar;<path to servlet-api.jar>\servlet-api.jar;
com.openmarket.gatorbulk.objects.BulkApprover config=bulkapprover.ini action=<notify|
approve|mark_publish> output_file=<out.txt>  target_list=Dynamic;;;;;;testdest
username=<adminUserName(fwadmin)> password=<adminUserPassword(xceladmin)>
```

Part VIII

Security: Managing Content Management Users

WebCenter Sites provides tags for authentication and user profile management and ACLs to enforce security.

Topic:

*   Managing Users on the Management System

45

Managing Users on the Management System

WebCenter Sites provides authentication functionality through the `USER` tags, user profile management through the `DIR` tags, and enforces security on database tables and rendered pages through access control lists (ACLs). You use these user management and security mechanisms to manage users and control user access on your distribution system and on your WebCenter Sites development and management systems.

Topics:

- About the Directory Services API

- Working with Custom User Management

- Controlling User Access

45.1 About the Directory Services API

The Directory Services API enables your WebCenter Sites system to connect to directory servers that contain authentication information, user information, and so on. WebCenter Sites delivers three directory services plug-ins, one of which is installed along with your WebCenter Sites systems.

- The WebCenter Sites directory services plug-in, which uses the native WebCenter Sites user management tables; that is, the `SystemUsers` and `SystemUserAttrs` tables.

- The LDAP plug-in, which supports any JNDI server.

- The NT plug-in, which retrieves user credentials and login name from the NT directory but gets all other user information from the `SystemUserAttrs` table.

The plug-in is installed during the installation of your WebCenter Sites systems and it is configured by setting properties in the `wcs_properties.json` file. For information about configuring your user management setup, see Understanding the LDAP Plug-In in *Oracle Fusion Middleware Administering Oracle WebCenter Sites*.

This section includes the following topics:

- Entries

- Hierarchies

- Groups

- Directory Services Tags

- Directory Operations

- [Error Handling](#)

- [Directory Services Applications Troubleshooting](#)

45.1.1 Entries

A directory entry is a named object with assigned attributes, in particular, user and group type entries:

- A user type object has a distinguished name and a set of attributes such as commonname, user name, password and email.

- A group type object, similar to a WebCenter Sites ACL, also has a distinguished name and a set of attributes.

Names reflect the hierarchy in which they are associated. To ensure portability across directory implementations, names should be treated as opaque strings.

45.1.2 Hierarchies

Some directory databases organize entries using a hierarchical structure. With WebCenter Sites directory services API, an entry's attributes and its place in the hierarchy are distinct. As a result, retrieving an entry's attributes does not yield information about its children.

Support for hierarchies depends on the underlying directory implementation; for example, LDAP directories support a hierarchical structure, while WebCenter Sites native directory database does not support a hierarchical structure. To ensure portability across directory implementations, your code should not assume support for hierarchical data.

> **Note:**
>
> Group hierarchies do not affect internal WebCenter Sites permissions.

45.1.3 Groups

WebCenter Sites directory services API does not enforce referential integrity. When you delete a user with the directory tags first remove the user from the groups that he is associated with. This ensure that group memberships are also deleted.

When a member is added to a group, the JNDI implementation always builds a fully distinguished name for the value of the `uniquemember` attribute, regardless of the name passed into the `addmember` tag.

45.1.4 Directory Services Tags

You can use the `DIR` tag family, as shown in the following table, with both XML and JSP versions to invoke the Directory Services API.

Table 45-1   Directory Services Tags

Tag	Description
DIR.ADDATTRS dir:addattrs	Adds attributes to an existing entry (which can be either a user or a group).

Table 45-1   (Cont.) Directory Services Tags

Tag	Description
DIR.ADDGROUPMEMBER dir:addgroupmember	Adds a member to a group (usually a user).
DIR.CHILDREN dir:children	Retrieves the child entries for a specified parent in a list variable.
DIR.CREATE dir:create	Creates a directory entry.
DIR.DELETE dir:delete	Deletes a directory entry.
DIR.GETATTRS dir:getattrs	Gets the attribute values for a specified entry in a list variable.
DIR.GROUPMEMBERS dir:groupmembers	Lists the members of a specified group.
DIR.GROUPMEMBERSHIPS dir:groupmemberships	Lists all the groups that an entry (either a group or a user) belongs to.
DIR.LISTUSERS dir:listusers	Returns a list of all the users in the directory.
DIR.REMOVEATTRS dir:removeattrs	Deletes an attribute value for an entry.
DIR.REMOVEGROUPMEMBER dir:removegroupmember	Removes an entry from a group.
DIR.REPLACEATTRS dir.replaceattrs	Replaces the value of an attribute for an entry (either a user or a group).
DIR.SEARCH dir:search	Searches the directory for entries who match the specified search criteria.

Regardless of whether the directory is implemented with LDAP or WebCenter Sites only, the code you write with the DIR tags is very similar.

See the *Oracle Fusion Middleware Tag Reference for Oracle WebCenter Sites Reference* and Directory Services Code Samples.

45.1.5 Directory Operations

Some WebCenter Sites Directory Services tags write information to the database. If your database administrators will be handling all of the website's write operations, such as adding user information to the database, restrict use of the directory tags to read-only operations. This policy avoids synchronization issues with third-party directory administration tools.

The read-only operations are presented in this section. These operations are performed using the credentials and read permissions of the currently authenticated user.

This section includes the following topics:

- Searching
- Looking Up a User
- Listing Users
- Directory Services Code Samples

45.1.5.1 Searching

Due to limitations in some directory servers, search is not allowed from the top organizational level. To avoid portability issues, always specify the context attribute on the `DIR.SEARCH` tag.

45.1.5.2 Looking Up a User

Looking up a user generally involves two steps:

1. Call `DIR.SEARCH` on the `userid` to get the entry name.

2. Call `DIR.GETATTRS` to get the attributes of the user in question.

45.1.5.3 Listing Users

It is recommended that you use one of the following three methods to list users:

- For small user databases, use the `DIR.LISTUSERS` tag, which recursively lists all users under the `peopleParent` property. This tag is inefficient on large user databases.

- For large user databases, use the `DIR.CHILDREN` tag to walk the hierarchy. The `DIR.CHILDREN` tag is best used for group types and not for user types.

- For user databases with a flat hierarchy, narrow results with a search.

45.1.5.4 Directory Services Code Samples

The following JSP code sample illustrates some possible directory operations:

```
<%
String sMainTestUserName = "user name";
String sMainTestUserPW="password";

String sPeopleParent = ics.GetProperty("peopleparent", "dir.ini", true);
String sGroupParent = ics.GetProperty("groupparent", "dir.ini", true);
String sUsername = ics.GetProperty("username", "dir.ini", true);
String sCommonName = ics.GetProperty("cn", "dir.ini", true);
IList mylist;
%>

<user:su username='<%=sMainTestUserName%>' password='<%=sMainTestUserPW%>'>

<H2>List All Users</H2>

<ics:clearerrno/>
<dir:listusers list='mylist'/>
<br>
<b>dir:listusers errno: <ics:getvar name='errno'/></b>
<ics:listloop listname='mylist'>
<br><ics:listget listname='mylist' fieldname='NAME'/>
```

```
</ics:listloop>

<H2>Look Up the ContentServer User by Username</H2>

<ics:clearerrno/>
<dir:search list='mylist' context='<%=sPeopleParent%>'>
<dir:argument name='<%=sUsername%>' value='ContentServer'/>
</dir:search>
<br><b>dir:search errno: <ics:getvar name='errno'/></b>

<%
mylist = ics.GetList("mylist");
if(mylist.numRows() != 1) {
out.print("<br>Error finding entry.");
}
mylist.moveTo(1);
ics.SetVar("ContentServerDn", mylist.getValue("NAME"));
%>

<H2>Show ContentServer Attributes</H2>

<ics:clearerrno/>
<dir:getattrs list='mylist'
name='<%=ics.GetVar("ContentServerDn")%>'/>
<br><b>dir:getattrs errno: <ics:getvar name='errno'/></b>
<ics:listloop listname='mylist'>
<br>
<ics:listget listname='mylist' fieldname='NAME'/>=
<ics:listget listname='mylist' fieldname='VALUE'/>
</ics:listloop>

<H2>Show Group Memberships for ContentServer</H2>

<ics:clearerrno/>
<dir:groupmemberships name='<%=ics.GetVar("ContentServerDn")%>'
list='mylist'/>
<br><b>dir:groupmemberships errno: <ics:getvar name='errno'/></b>
<ics:listloop listname='mylist'>
<br>
<ics:listget listname='mylist' fieldname='NAME'/>
</ics:listloop>

<H2>Lookup the SiteGod Group by CommonName</H2>

<ics:clearerrno/>
<dir:search list='mylist' context='<%=sGroupParent%>'>
<dir:argument name='<%=sCommonName%>' value='SiteGod'/>
</dir:search>
<br><b>dir:search errno: <ics:getvar name='errno'/></b>

<%
mylist = ics.GetList("mylist");
if(mylist.numRows() != 1) {
    out.print("<br>Error finding entry.");
}
mylist.moveTo(1);
ics.SetVar("SiteGodDn", mylist.getValue("NAME"));
%>

<H2>Show SiteGod Attributes</H2>
```

```
<ics:clearerrno/>
<dir:getattrs list='mylist' name='<%=ics.GetVar("SiteGodDn")%>'/>
<br>
<b>dir:getattrs errno: <ics:getvar name='errno'/></b>
<ics:listloop listname='mylist'>
<br>
<ics:listget listname='mylist' fieldname='NAME'/>=
<ics:listget listname='mylist' fieldname='VALUE'/>
</ics:listloop>

<H2>Show SiteGod Group Members</H2>

<ics:clearerrno/>
<dir:groupmembers name='<%=ics.GetVar("SiteGodDn")%>' list='mylist2'/>
<br>
<b>dir:groupmembers errno: <ics:getvar name='errno'/></b>
<ics:listloop listname='mylist2'>
<br>
<ics:listget listname='mylist2' fieldname='NAME'/>
</ics:listloop>

<H2>Children of groupparent </H2>

<ics:clearerrno/>
<dir:children name='<%=sGroupParent%>' list='mylist'/>
<br>
<b>dir:children errno: <ics:getvar name='errno'/></b>
<ics:listloop listname='mylist'>
<br>
<ics:listget listname='mylist' fieldname='NAME'/>
</ics:listloop>

</user:su>
```

45.1.6 Error Handling

Any of the directory tags can cause a range of directory errors to be set. See the *Oracle Fusion Middleware Tag Reference for Oracle WebCenter Sites Reference* for a comprehensive list of directory services error messages.

Your directory services code should handle every one of the error codes listed for a given tag call. This is necessary to support the J2EE JNDI interface.

45.1.7 Directory Services Applications Troubleshooting

The first step in troubleshooting directory services applications is to check the error log (in the WebCenter Sites log file).

You enable directory services logging by setting the log.filterLevel property (found in the logging.ini property file). There are seven levels of error messages that you can view:

- fatal: Logs fatal level messages.

- severe: Logs severe and fatal level messages.

- error: Logs error and fatal level messages.

- warning: Logs warning and fatal level messages.

- info: Logs warning, error, severe, and fatal level messages.

- trace: Logs trace messages.

- detail: Logs all types of messages.

During troubleshooting, `trace` is the most verbose setting, and as a result, has the highest performance impact.

Directory services log entries use the following format:

```
[<timestamp>][Directory-<severity>-<errno>]
[<class>:<method>][<message>][<session id>]
```

For example:

```
[Jan 17, 2002 1:49:44 PM][Directory-T]
[BaseFactory:instantiateImplementation(ICS,String,Class[],
Object[])][Instantiating:com.openmarket.directory.common.Factory]
[PEccxyFlUeh7zYvjNgg4D6bqZzf0llfWMaiBimIN9HlZ9KomDcPy]
```

The previous message is a trace (T), and thus has no associated `errno` value.

See Logging and Debugging Errors.

A common problem for LDAP implementations is incorrectly specified permissions on the directory server. If the error log indicates a permission problem, ensure that the authenticated user has permissions to execute the requested operation by checking the permission settings on the directory server. Try logging into the directory server directly (outside of WebCenter Sites) and performing the same action to ensure that permissions are correctly set. After checking the log and permissions, you can often resolve a configuration error by examining the property files.

See the *Oracle Fusion Middleware Property Files Reference for Oracle WebCenter Sites*.

45.2 Working with Custom User Manager

You can access user information stored in the database tables or in LDAP (or other Directory protocols) using the implementation available through the CustomUserManager class.

Topics:

- What is Custom User Manager?

- Sample Implementation of Custom User Manager

- Integrating the Sample Implementation with WebCenter Sites

- What You May Need to Know About the Custom User Manager

45.2.1 What is Custom User Manager?

For those who want greater flexibility than that provided by the WebCenter Sites out of the box authentication mechanism, Oracle also provides implementations to access user information stored in the database tables or in LDAP (or other Directory protocols). These implementations are available through the CustomUserManager class.

The CustomUserManager class enables clients to implement a connection to an arbitrary user repository by extending the existing architecture. This connection is used to authenticate and authorize WebCenter Sites users. It provides read-only access to the user directory. The write-access is not required because all user maintenance operations are performed in a central directory system. Those who implement

UserDirectory are responsible for correctly mapping the user attributes of the users in the arbitrary user repository to those that WebCenter Sites uses. For example, ACLs and Roles per site.

Interfaces to Extend a Site Architecture

- `oracle.fatwire.sites.directory.custom.UserDirectory`: Clients implement this interface. WebCenter Sites invokes the client's implementation to authenticate the user and obtain user roles and ACLs.

- `oracle.fatwire.sites.directory.custom.UserFactory` and `oracle.fatwire.sites.directory.custom.UserFactory.Builder`: WebCenter Sites provides an implementation of these interfaces as a container to populate user information.

See *Oracle Fusion Middleware Java API Reference for Oracle WebCenter Sites*.

Webcenter Sites doesn't cache any user information or roles from the arbitrary user repository. Clients can customize their implementation to handle caching of user information and roles. Clients can create new roles in WebCenter Sites after setting this property in the `wcs_properties.json` under the `config` directory:
`xcelerate.rolemanagerclass=com.openmarket.xcelerate.roles.RoleManager`

These roles can't be updated.

All property changes are retained when WebCenter Sites is upgraded. However, clients need to back up any classes used for the UserDirectory implementation.

Logger for Custom User Manager

A new logger `oracle.wcsites.directory.custom` is added for this Custom User Management customization hook. This hook provides clients with the flexibility to customize or define their own authentication. Use log level TRACE. Legacy code continues using old loggers (`dirLogger/ics.LogMsg()` and `logging.ini`, `com.fatwire.logging.cs.auth`, `com.fatwire.logging.cs.session` etc.).

45.2.2 Sample Implementation of Custom User Manager

To help you develop a good understanding of Custom User Manager, a sample implementation is provided in the WebCenter Sites repository.

The sample code is located under `<ORACLE_HOME>/wcsites/webcentersites/sites-home/bootstrap/samples/CustomUserManager`.

This implementation consists of the following files:

- `user-repository.json`: This json file stores the user details such as username, encrypted password, roles, and ACLs. Ensure that the userId contains the proper parent string as it was defined during installation. See the peopleparent property in the `wcs_properties.json` file for the value of the parent string.

- `FileUserRepository.java`: It is a backend store for user information. It reads the json file and populates the SampleUser objects in the memory. You need to restart the server to reflect any changes to `user-repository.json`.

- `SampleUser.java`: Holds the user information in the memory.

- `SampleUserRepository.java`: This class implements the `oracle.fatwire.sites.directory.custom.UserDirectory` interface.

WebCenter Sites invokes this implementation when it needs to authenticate a user, get the user roles, list users matching criteria and so on.

The sample implementation project is easy to compile with WebCenter Sites 12.2.1.2.0 and above. To understand the project setup, see ReadMe.txt under `<ORACLE_HOME>/wcsites/webcentersites/sites-home/bootstrap/samples/CustomUserManager`.

45.2.3 Integrating the Sample Implementation with WebCenter Sites

All you need to do to integrate the sample implementation of Custom User Manager is change and add a few properties to the `wcs_properties.json` file, compile the project, and update the application server class path.

1. Change the following properties in the `wcs_properties.json` file:

   - `cs.manageUser=oracle.fatwire.sites.directory.custom.CustomLogin`

   - `cs.manageproperty=dir.ini`

   - `xcelerate.usermanagerclass=oracle.fatwire.sites.directory.custom.CustomUserManager`

   - `className.IDir=oracle.fatwire.sites.directory.custom.CustomDir`

   - `className.IUserDir=oracle.fatwire.sites.directory.custom.CustomDir`

   - `xcelerate.rolemanagerclass=com.openmarket.xcelerate.roles.RoleManager`

2. Add the following properties to the `wcs_properties.json` file:

   - `className.UserDirectory=oracle.fatwire.sites.auth.sample.SampleUserRepository`

     If this property does not exist, add:

   - `defaultReaderACLs=Browser,Visitor`

   Of all the properties that need to be changed, only the `className.UserDirectory` is specific to each client. The others are needed to make use of this plugin. The `className.UserDirectory` property needs to contain the class name of the custom implementation of the UserRepository.

3. Compile the project and add the classes to the WebCenter Sites webapp `WEB-INF/lib` folder.

4. Place the `user-repository.json` file in the application server's classpath so the `FielUserRepository.java` file can read the users.

5. To verify:

   a. Install WebCenter Sites (with or without LDAP), and test if it is working.

   b. Change and add the properties discussed in step 1 and 2.

   c. Add the `CustomUserManager-sample-0.0.1-SNAPSHOT.jar` and the `user-repository.json` file to the application server's class path.

     d. Restart and log into the application server with the credentials given in the json file.

45.2.4 What You May Need to Know About the Custom User Manager

The UserDirectory interface provides a read-only access to the user directory because all user maintenance operations are performed in a central directory system.

- As a result, the following operations may not work or show erroneous success messages:

    - The <DIR> tags allow you to perform functions such as creating and updating user profiles and adding user roles. When these tags are invoked, WebCenter Sites logs the operation not supported exception and sets the error number -15004 in the ics scope.

    - WebCenter Sites UI uses these tags in many places. UI forms have not been changed to reflect the error message on the interface pages. As a result, updating a user profile (or other such operations) may erroneously indicate the operation was successful but in reality nothing changes.

- REST Groups security must work as before. When accessing the REST resources make sure that the users have been assigned to proper groups.

45.3 Controlling User Access

WebCenter Sites manages users through access control lists (ACLs). By using ACLs, you can restrict access to tables in the WebCenter Sites database and the rendered pages served on your sites by WebCenter Sites. You must associate registered users with one or more ACLs for a site to which users log in with user names and passwords.

When a user first visits a site, WebCenter Sites creates a session and implicitly logs in the user as the standard default user, `DefaultReader`. The identity of a user is updated (and any associated ACLs go into effect) when a `USER.LOGIN` command is used and the user is authenticated against a password.

See these topics:

- ACL Tags

- USER Tags

- WebCenter Sites and Encryption

45.3.1 ACL Tags

WebCenter Sites provides a set of access control list tags (both XML and JSP versions) that you can use to create ACLs. You can use either the WebCenter Sites interface on the management system or the WebCenter Sites ACL tags to create the ACLs that you need for your user accounts on your management system. The following table lists the ACL tags:

Table 45-2   ACL Tags

Tag	Description
ACL.CREATE acl: create	Creates an ACL.

Table 45-2   (Cont.) ACL Tags

Tag	Description
`ACL.DELETE acl:delete`	Deletes an ACL.
`ACL.GATHER acl:gather`	Gathers fields into an ACL.
`ACL.GET acl:get`	Copies a field from an ACL.
`ACL.LIST acl:list`	Retrieves a list of ACLs.
`ACL.LOAD acl:load`	Loads an ACL.
`ACL.SAVE acl:save`	Saves an ACL.
`ACL.SCATTER acl:scatter`	Scatters a field from an ACL.
`ACL.SET acl:set`	Sets a field in an ACL.

For more information:

- ACL tags: See the *Oracle Fusion Middleware Tag Reference for Oracle WebCenter Sites Reference*.

- ACLs in general: See *Oracle Fusion Middleware Administering Oracle WebCenter Sites*.

45.3.2 USER Tags

WebCenter Sites also provides the USER tags (both XML and JSP versions) described in the following table. You use these tags on pages that log users in and out.

Table 45-3   User Tags

Tag	Description
USER.LOGIN `user:login`	Logs a user in.
USER.LOGOUT `user:logout`	Logs a user out.
USER.SU `user:su`	Logs the user in as a specific user to perform an operation such as creating an account or edit a user profile.

See the *Oracle Fusion Middleware Tag Reference for Oracle WebCenter Sites Reference*.

45.3.3 WebCenter Sites and Encryption

WebCenter Sites includes a default key for encrypting passwords and other sensitive information. You can specify your own encryption key by using the `Utilities` class `encryptString` method. See the *Oracle Fusion Middleware Java API Reference for Oracle WebCenter Sites* for information about Java methods that deal with encryption.

WebCenter Sites also supports Secure Sockets Layer (SSL), which allows encryption of information going to and from your web servers. See Implementing Security in *Oracle Fusion Middleware Administering Oracle WebCenter Sites*.

Part IX

Publishing Your Site

WebCenter Sites provides these publishing methods: RealTime, Mirror to Server, Export to Disk, and export to XML.

Topic:

- Publishing Your Content Management Site to Make it Available Online

46

Publishing Your Content Management Site to Make it Available Online

The publishing system provides means to migrate sites and their content from one system to another. WebCenter Sites provides these publishing methods: RealTime, Mirror to Server, Export to Disk, and export to XML. You use the Mirror to Server or RealTime publishing method to migrate the structure of the site (that is, the database schema).

For information about using different publishing methods, see these topics in *Oracle Fusion Middleware Administering Oracle WebCenter Sites*:

- Understanding Publishing

- Working with Export to Disk Publishing

- Working with Mirror to Server Publishing and Export to XML

- Exporting Assets to XML Publishing Method

- Using RealTime Publishing
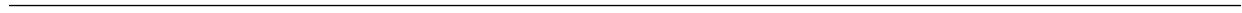
Part X

Developing Personalized and Targeted Websites with Engage

Visitor data assets let you group your site visitors into segments. Recommendation assets collect, evaluate, and sort product and content assets to recommend the most appropriate flex asset. The memory-centric method enables you to track visitors. There are some requirements that you need to address before you implement this method. With Engage you can design an online site that gathers visitor information and personalize promotional messages for each visitor.

Topics:

- Creating Visitor Data Assets

- Understanding Recommendation Assets

- Working with Memory-Centric Visitor Tracking

- Coding Engage Pages

47

Creating Visitor Data Assets

Oracle WebCenter Sites: Engage lets you design online sites that gather information about your site visitors and customers, and then use that information to personalize the product placements and promotional offerings for each visitor. The information about visitors is stored in visitor data assets. There are three kinds of visitor data assets: visitor attributes, history attributes, and history types. The definitions of visitor data types are treated as assets in the WebCenter Sites database.

Topics:

- About Visitor Data Assets

- Creating Visitor Data Assets

- Verifying Visitor Data Assets

- Approving Visitor Data Assets

47.1 About Visitor Data Assets

Do you want to group your site visitors into segments? You can do this with the three types of visitor data assets: Visitor attributes, History attributes, and History types.

To create visitor data assets, you create entries in the visitor data tables in the WebCenter Sites database and reserve a place in the database to store information of that kind for your site visitors.

See these topics:

- Visitor Attributes

- History Attributes and History Definitions

- Segments

- Developing Visitor Data Assets: Process Overview

47.1.1 Visitor Attributes

Visitor attributes hold types of information that specify one characteristic only (scalar values). For example, you can create visitor attributes named years of experience, job description, or number of children.

When the visitor changes the data, the new data overwrites the old data. Engage does not assign a timestamp to the data that is stored as a visitor attribute and does not store revisions. For example, if a visitor changes his entry for job description from butcher to baker, the information that the visitor was once a butcher is overwritten. You cannot, for example, create a segment based on bakers who used to be butchers.

For historical data, you must use history types.

47.1.2 History Attributes and History Definitions

History attributes are individual information types that you group together to create a vector of information that Engage treats as a single record. This record is the **history definition**. For example, a history definition called purchases can consist of the history attributes SKU, itemname, quantity, and price.

Engage references data stored as a history definition as a whole or an aggregate. It assigns a timestamp to each instance of the recorded definition and keeps each of those records. This means that you can sum or count history definitions and you can determine the first time or the last time a history definition was recorded for a visitor. Using the example in the preceding paragraph, you can create a segment based on the amount of money a visitor spends on specific items during a set period of time.

History definitions store historical data.

47.1.3 Segments

Segments are assets that divide visitors into groups based on common characteristics. Segments are built by determining which visitor data assets to base them on and then setting qualifying values for those criteria.

To create visitor data assets, you create fields that can be used in two places:

- As criteria for segments. That is, as configuration options in the Engage Segment Filtering forms (because you define segments with the visitor data assets). In other words, the choices you make about the data types for the attributes determine their appearance and behavior in the Segment forms. When you create these assets, you are customizing the Segment forms.

- On your public site pages. That is, as form fields or hidden fields on registration pages and other pages.

Segments are the key to personalizing merchandising messages with Engage. When visitors browse your site, the information they submit is used to qualify them for segment membership. When the site opens a page with a recommendation or promotion, Engage determines which segments a visitor belongs to and opens the product recommendations or promotional messages that are designated for those segments.

Configuring Bluekai Type of Segments

Websites that use Bluekai (Oracle Marketing Data Cloud) can track site visitors using the Bluekai related WebCenter Sites APIs. WebCenter Sites provides two types of APIs that help identify visitor behavior and recommend personalized content respectively.

Bluekai Client Side API: This JavaScript API helps identify the visitor behavior. It is available at `<sites-war>/integration/bluekai/bluekaihelper.js`. This API uses a JavaScript object whose constructor contains the following signature:

```
function(bluekaiSitesCookieName, bkSegmentsCookiePrefix, pixelLimit, expires)
```

where:

- `bluekaiSitesCookieName`: Value of the `bluekai.sites.cookie` property in WebCenter Sites.

- `bkSegmentsCookiePrefix`: Value of the `bluekai.segments.cookie.prefix` property in WebCenter Sites.

- `pixelLimit` – It's a Bluekai property. It indicates the number of slots available in the container for firing image tags. (See Integrating into the Oracle Bluekai Platform in *Using Oracle Data Cloud*) If you do not provide any value for this parameter in the constructor, API sets its default value as 1.

- `expires` : A numeric value indicating the cookie expiry time as number of days. This expiry time is for the cookie which this JavaScript API creates. To address the case when a user does not want to store this cookie for a fixed period (that is, the cookie should remain only for the current session), the value for this property should be zero (0). The default value is 1.

The JavaScript object contains these methods:

`fetchSements(siteID)`: Based on information from Bluekai, this method determines WebCenter Sites segments for which the current visitor qualifies. Client side developers should use this API in web pages (preferably in a wrapper) of their site as follows:

```
<html> <html>
<head>
<script type="text/javascript" src="http://<server host n port>/sites/js/
integrations/bluekai/bluekaihelper.js"></script>
</head>
<body>
<script>
var bkClient = new com.oracle.bluekai.Client('<
%=ics.GetProperty("bluekai.sites.cookie")%>','<
%=ics.GetProperty("bluekai.segments.cookie.prefix")%>',1,2); // expiry time is 2
days for cookies to be created using call 'fetchSegments'
bkClient.fetchSegments(bluekaiContainerID); //'bluekaiContainerID' is the site ID
created in the Bluekai partner website.
</script>
....
...
</html>
```

Assuming that some segments of the Bluekai type have been created in WebCenter Sites, the above code identifies those Segment assets that the current site visitor qualifies for, based on the data (campaigns) returned by Bluekai. This API then sets these segment names (comma separated) in a cookie. The naming convention of the cookie is as follows:

`<bluekai.segments.cookie.prefix><bluekaiContainerID>`

For example, if the value of the `bluekai.segments.cookie.prefix` property is `extSegments` and `bluekaiContainerID`'s value is `'55552'`, then the cookie name would be `'extSegments55552'`. And, its value would be the names of the Bluekai type of Segments. If no segments are qualified (either because Bluekai knows nothing about current visitor OR WebCenter Sites has no Bluekai Segments matching campaigns returned by Bluekai), then the cookie value would be 'none'. Before the cookie expires, the subsequent call to the `fetchSegments` method does not hit Bluekai anymore. This saves money, as each call to Bluekai incurs cost.

`postUserData(siteId, data)`: Websites can use this Java Script API to post the visitor behavior data to Bluekai on behalf of the current site visitor. For example, if a website is dedicated to tennis fans, it can let the API know that the website visitors are interested in the 'Tennis' sport. To inform the API, the website can post a key-value pair to Bluekai indicating that the current visitor is interested in tennis, as follows:

```
var data = []; data["interest"]="Tennis"; data["location"] =
"India"; bkClient.postUserData(<siteID>,data);
```

In Bluekai terminology, these pieces of information are called pHints. You should create 'rules' in the Bluekai partner site, say interest is Tennis. You use these rules in a category and use that category while creating Audience to associate a campaign with that audience. Once this is done, the above code sends data to Bluekai that enables the current site visitor to qualify for this campaign.

Server Side API: The Java API BluekaiHelper (available in the package oracle.fatwire.integrations.bluekai) provides the names of qualified bluekai segments by reading the cookie created above. This API should be used in conjunction with WebCenter Sites Engage API to display recommended items to a site visitor. Class BluekaiHelper includes the getQualifiedBluekaiSegments(ICS) method, which can be used to get recommended assets. For example, you create a segment named interestedInTravel which uses a campaign created in Bluekai, meant for identifying visitors interested in travelling. You create a recommendation asset, say named TravelRecommendation which uses this segment to define what all articles/assets should be shown to a visitor that qualifies for the interestedInTravel segment. So, the server side code to fetch recommended articles for current visitor (this sample code is assumed to be in a template controller) can be as follows:

```
List<Map> recommendations = newRecommendationReader()
                            .forSegments(BluekaiHelper.getInstance().getQualifiedBlue
kaiSegments(ics) )
                            .forSite("avisports")
                            .readRecommendations("TravelRecommendation");
```

47.1.4 Developing Visitor Data Assets: Process Overview

There are five general steps for creating and using visitor data assets (fields):

1. A cross-functional design team including developers and marketers determines what kind of data you want to gather about your site visitors.

2. You (the developers) create and define the necessary visitor attributes, history attributes, and history definitions by using the forms in Engage.

3. The marketers use the Segment Filtering forms in Engage to categorize groups of visitors based on these visitor attributes, history attributes, and history definitions.

4. You program the appropriate site pages with the Engage XML or JSP object methods to collect and store the data, using either server-side validation or Javascript to validate the input on the pages. For example, you can create an online registration form for visitors to fill out by using JavaScript to validate the input and the Engage XML or JSP tags to process and store that information in the WebCenter Sites database.

5. When visitors browse your site, the information they submit is used to qualify them for segment membership. If your site is using promotions and recommendations based on segments, the message displayed for the visitor is personalized based on the segments that he or she qualifies for.

47.2 Creating Visitor Data Assets

Before you begin creating visitor data assets, meet with the marketing and design teams to determine the kinds of data that you want to collect about visitors. And,

examine the Segment Filtering forms to understand the context in which the visitor data assets that you create are used by the marketers.

Additionally, note that the **visitor data assets** are listed by their **descriptions** rather than by their names in the Segment Filtering forms.

You can use the following data definitions for your visitor and history attributes:

> **Note:** When you're using the `vdm:setscalar` API, keep in mind that it doesn't validate the size limit that you've specified for each data type. Therefore, Oracle recommends that you follow these ranges when you're specifying values for scalar attributes for each type.

- String: Can hold up to 255 characters.

- Boolean: True and false are the only legal values.

- Short: Valid range of values is 0 through 255.

- Integer: Valid range of values is 0 through 65,535.

- Long: Valid range of values is 0 through 65,535.

- Double: Valid range of values is 0 through 4,294,967,295.

- Date: format is `yyyy-mm-dd hh:mm:ss.s`

- Money: Format is currency; valid range of values is unlimited.

- Binary: For visitor attributes only; used for binary data such as image files or cart objects.

> **Note:**
>
> Binary visitor attributes can record binary data for individual visitors. Visitor attributes of this type are not displayed in the Segment Filtering forms and cannot be used to define a segment. Creating an attribute of type binary reserves space in the WebCenter Sites database that you use to store objects by using the XML object method `VDM.SAVESCALAROBJECT` or its JSP equivalent `vdm:savescalarobject` to convert an object from the WebCenter Sites name space into a binary form.

This section includes the following topics:

- Creating Visitor Attributes

- Creating History Attributes

- Creating History Definitions

47.2.1 Creating Visitor Attributes

Create visitor attributes with the Engage forms.
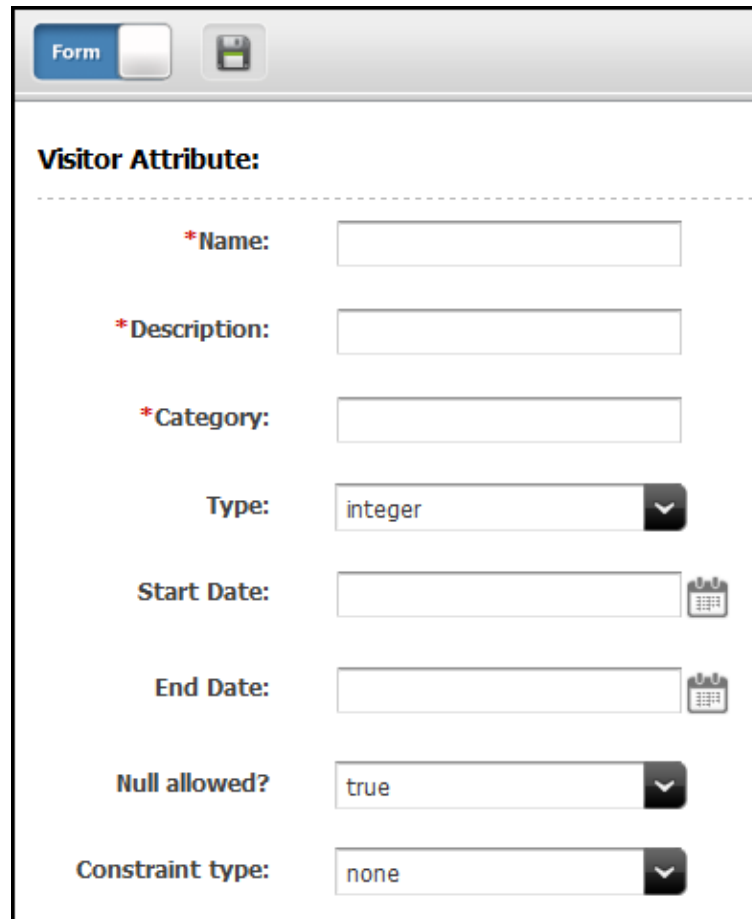
To create a new visitor attribute:

1. Open the Admin interface.

2. Click **New** on the button bar.

3. From the list of asset types, select **New Visitor Attribute**. (Site Visitor Attribute asset types must be enabled for your site.)

The Site Visitor Attribute form opens.

Figure 47-1   Site Visitor Attribute

> **Note:**
>
> **Visitor Attribute** does not appear in the menu when your login password combination doesn't have administrator rights. Contact the site administrator and request that the admin user profile be assigned to your user name.

4. In the **Name** field, enter a unique, descriptive name for the attribute (field). You can enter up to 32 alphanumeric characters, including spaces. The first character must be a letter.

5. In the **Description** field, enter a description of the attribute (field). Enter a value (alphanumeric characters) that help you easily identify the attribute (attributes are listed by their descriptions rather than their names in the Segment Filtering forms).

6. In the **Category** field, enter the category for the attribute. The text that you enter in this field determines where the attribute is listed in the Segment Filtering form. You can enter up to 32 alphanumeric characters.

> **Note:**
>
> Categories for visitor attributes must be different from the categories for history definitions.

47.2.1.1 Configure the Data Type

To configure the data type:

1. From the **Type** drop-down list, choose a data type.

2. If you chose **string**, then in the **Length** field enter the maximum number of characters allowed for input in the attribute (field). You can enter a value up to 255.

3. From the **Null allowed** drop-down list, choose **true** to allow null values or **false** to require input for the attribute when it is used. For example, an attribute with a Boolean data type cannot allow a null value.

4. If you chose **false** from the **Null allowed** drop-down list, then in the **Default Value** field enter a default value that is appropriate for the attribute's data type. For example, for the integer datatype, the default value must be a number.

> **Note:**
>
> If you selected **binary** as the data type, you cannot specify a default value for the attribute.

47.2.1.2 Configure the Constraint Criteria

The constraint options that are available for validating input into the attribute depend on the data type that you designated for the attribute.

Option 1: Configure the Attribute to Accept Free-Form Text

From the **Constraint type** drop-down list, choose **none**. For example, a visitor attribute named `residence` of type `string` might accept unconstrained text as input.

Option 2: Configure the Attribute to Accept Input from a Range of Values

To configure the attribute to accept a specific range of values, the data type must be integer, short, long, double, or money.

1. From the **Constraint type** drop-down list, choose **range**.

   The form displays range fields.

2. In the **Lower range limit** field, specify the smallest possible value that can be accepted in the attribute when it is used as a field. This value cannot be a negative number.

3. In the **Upper range limit** field, enter the largest possible value that can be accepted in the attribute when it is used as a field. (For a short data type, you can enter a value up to 255; for integer, up to 65,535; for double, up to 4,294,967,295; for money, unlimited.)

   For example, an attribute named `Age` can be restricted to accept values between 1 and 110 only.

Option 3: Configure the Attribute to Offer a Set List of Values in a Drop-Down List

1. From the **Constraint type** drop-down list, choose **enumeration**.

   The form displays text boxes for adding options.

2. In the **Add Enumerated Value** field, enter the name of the first option. For example, an attribute named `gender` can have `female` as an option.

3. Click **Add**.

   The option is moved to the list.

   Figure 47-2   Enumeration Constraints

4. Repeat these steps for each of the options that you want to make available for this attribute (field).

47.2.1.3 Save the Attribute

1. (Optional) If you have access to multiple sites, specify whether you want to share this attribute with them. On the Inspect form, from the **More** menu, choose **Share Visitor Attribute**.

2. When you are finished configuring the visitor attribute, click **Save**.

   Engage creates an entry for this attribute in the visitor data asset tables in the WebCenter Sites database and reserves a place in the database to store information of that type for your site visitors.

   Engage opens a summary of the attribute in the Inspect form.

You can now use this visitor attribute in a segment.

> **Note:**
>
> After a visitor attribute is used to define a segment, deleting the attribute invalidates the segment. Be sure to correct your segments if you delete an attribute.

47.2.2 Creating History Attributes

The purpose of history attributes is different from the purpose of visitor attributes: you create history attributes to be used by history definitions. You cannot use them in the Segment Filtering forms until they are used to define a history definition.

Perform the procedures in this section to create history attributes by using the Engage forms.

Note:

You cannot edit or delete a history attribute after it has been used to define a history definition. You also cannot remove it from the history definition. Therefore, stop using the history definition whose history attribute has been changed. Create a new history attribute, create a new history definition, and then start using the new history definition.

To create a new history attribute:

1. Open the Admin interface.

2. On the button bar, click **New**, then select **New History Attribute** from the list.

   The History Attribute form opens.

Figure 47-3   History Attribute Form

Note:

If History Attribute does not appear in the menu, it means that your login/
password combination does not give you administrator rights. Contact the site
administrator and request that the admin user profile be assigned to your user
name.

3. In the **Name** field of the History Attribute form, enter a unique, descriptive name
   for the attribute (field). You can enter up to 32 alphanumeric characters, including
   spaces. The first character must be a letter.

4. In the **Description** field, enter a description of the attribute (field). Enter a value
   (alphanumeric characters) that help you easily identify the attribute (attributes are
   listed by their descriptions rather than their names in the Segment Filtering forms).

5. From the **Type** drop-down list, choose a data type.

6. If you selected **string**, in the **Length** field enter the maximum number of characters
   allowed for input in the attribute (field).

7. To make this attribute a required field when the history definitions that use it
   define a segment, select **true** in the **Must be specified** field.

8. From the **Filter** drop-down list, choose **true**.

   If you do not set **Filter by** to **true**, the marketers cannot use the attribute (field) as a constraint for any history definition that it belongs to when they create segments.

   If the data type for this attribute is numeric, then by default the attribute is included in the list of attributes that can be selected for a Total constraint in a segment, whether you set **Filter by** to true or to false. However, to use a numeric attribute as a constraint in any other way, you must set **Filter by** to true.

9. From the **Null allowed** drop-down list, choose **true** to allow null values or **false** to require input for the attribute when it is used. For example, an attribute with a Boolean data type cannot allow a null value.

10. If you chose **false** from the **Null allowed** drop-down list, in the **Default Value** field enter a default value that is appropriate for the attribute's data type. For example, the default value for the integer datatype must be a number.

47.2.2.1 Configure the Constraint Criteria

The constraint options available for validating input into the attribute depend on the data type you designated for the attribute.

Configure the Attribute to Accept Free-Form Text

From the **Constraint type** drop-down list, choose **none**. For example, a history attribute named `Street Name` of type `string` might accept unconstrained text as input.

Configure the Attribute to Accept Input from a Range of Values

To configure the attribute to accept a specific range of values, the data type must be integer, short, long, double, or money. This figure shows the range constraints and related fields.

Figure 47-4   Range Constraints and Related Fields

1. From the **Constraint type** drop-down list, choose **range**.

   The form displays range fields.

2. In the **Lower range limit** field, specify the smallest possible value that can be accepted in the attribute when it is used as a field. This value cannot be a negative number.

3. In the **Upper range limit** field, enter the largest possible value that can be accepted in the attribute when it is used as a field. (For a short data type, you can enter a value up to 255; for integer, up to 65,535; for double, up to 4,294,967,295; for money, unlimited.)

> For example, an attribute named Number of Items can be restricted to accept values between 1 and 50 only.

Configure the Attribute to Offer a Drop-Down List of Specific Values

1. From the **Constraint type** field, choose **enumeration**.

   The form displays text boxes for adding options.

   Figure 47-5   Enumeration Constraints Fields

2. In the **Add Enumerated Value** field, enter the name of the first option.

3. Click **Add**.

4. Repeat these steps for each of the options that you want to make available for this attribute.

47.2.2.2 Save the History Attribute

When you are finished configuring the history attribute, click **Save**.

Engage creates an entry for this attribute in the visitor data asset tables in the WebCenter Sites database and reserves a place in the database to store information of that type for your site visitors.

You can now use this history attribute to define a history definition.

47.2.3 Creating History Definitions

History definitions are made up of history attributes. Therefore, there must be at least one history attribute created before you can create a history definition.

To create history definitions using the Engage forms:

1. Log in to the Admin interface.

2. Click **New** and select **History Definition** from the list.

   The History Definition form opens.

Figure 47-6 History Definitions Fields

Note:

History Definition does not appear in the menu if your login/password combination does not have administrator rights. Contact the site administrator and request that the admin user profile be assigned to your user name.

3. In the **Name** field, enter a unique, descriptive name for the history definition (record). You can enter up to 32 alphanumeric characters, including spaces. The first character must be a letter.

4. In the **Description** field, enter a description of the history definition. Enter a value (alphanumeric characters) that help you easily identify the history definition (history definitions are listed by their descriptions rather than their names in the Segment Filtering forms).

5. In the **Category** field, enter a category for the history definition. The text that you enter in this field determines how the history definition is sorted and displayed in the Segment Filtering forms. You can enter up to 32 alphanumeric characters.

Note:

Categories for history definitions must be different from the categories for visitor attributes.

6. In the **Fields** area, select the history attributes that make up this history definition. Select an attribute and then click the right arrow to move it to the list on the right. Use **Ctrl+click** to select multiple attributes at the same time.

> **Note:**
>
> After a history attribute is used to define a history definition, you can no longer edit or delete that history attribute.

7. Click **Save**.

Engage creates an entry for this history definition (record) in the visitor data asset tables in the WebCenter Sites database and reserves a place in the database to store information of that type for your site visitors.

Engage then opens a summary of the history definition in the Inspect form.

You can now use this history definition in a segment.

47.3 Verifying Visitor Data Assets

To determine that the visitor attributes, history attributes, and history definitions are properly set up, examine the Segment Filtering forms and check whether the visitor assets that you created were configured correctly.

- Create segments that use each of the visitor attributes and history definitions that you created.

- Determine that the constraint definitions are correct and that the input ranges are accepting the correct range of input.

47.4 Approving Visitor Data Assets

To correctly publish a history definition, you must also approve its history attributes for publishing.

When your visitor data assets are ready, approve them so that they can be published to the delivery system.

To approve any asset, choose **Approve for Publish** from the drop-down list in the icon bar in the asset's Edit or Inspect form.

48

Understanding Recommendation Assets

You use Recommendations assets to determine which products or content should be featured or recommended on a rendered page. These assets are a set of rules that might be based on the segments the visitors qualify for, and, in some cases, relationships between the product or content assets or both.

These topics describe what recommendations are and how you develop them, and how you can create a custom element that returns recommendable assets:

- About Recommendation Assets

- Development Process for Setting Up Recommendations

- About Creating a Dynamic List Element

48.1 About Recommendation Assets

A recommendation asset collects, evaluates, and sorts product and content assets. It determine the most appropriate assets by consulting the list of segments that the visitor belongs to, and it recommends the chosen flex assets for the current visitor.

The product assets and content flex assets are rated for their importance to each segment. When a recommendation asset is called from a template, Oracle WebCenter Sites: Engage determines which segments the current visitor qualifies for, and then selects the assets that are identified by the recommendation as having the highest rating for those segments. These are the assets that are recommended to the visitor.

There are three kinds of recommendations:

- **Static Lists**: Return a static list of recommended items.

- **Dynamic Lists**: Return a list of recommended items that is generated by a dynamic list element that you create.

- **Related Items**: Return a list of recommended items based on relationships between flex assets, such as products.

Engage uses a recommendation's configuration options and the asset ratings to constrain the list when the list contains more items than the template is programmed to display. For related items recommendations, Engage also uses asset relationships to constrain the list. For all recommendations, Engage eliminates assets that are rated 0 for the current visitor.

To implement Insite editing for Recommendation assets, use `<insite:slotlist>`, but you need to use `field="Manualrecs"`, and it only works for Recommendations of type List.

The recommendation asset is the only Engage asset that can be assigned a template. You code your recommendation templates to render the items that the recommendation returns in an appropriate way on the rendered page.

The template tells the recommendation how many assets to return, and the recommendation asset determines which assets to select and return to the template based on the way it is configured and on the segments that the current visitor belongs to.

There are several XML and JSP object methods (tags) that you can use to code templates for recommendations. For information about coding templates when you are using Engage, see Coding Engage Pages. For information about all of the Engage tags see the *Oracle Fusion Middleware Tag Reference for Oracle WebCenter Sites Reference*.

48.2 Development Process for Setting Up Recommendations

Here are basic steps to help you understand how you can set up recommendations.

1. Developers and designers meet with the marketing team to define all the merchandising messages that you want to display on your site and to plan how to represent those messages using recommendation and promotion assets.

2. The developers and designers use the XML or JSP object methods to design and code templates for the recommendations. Coding Engage Pages explains how to code these templates.

3. If the website uses dynamic list recommendations, then the developers code the dynamic list elements that return the assets to recommend. About Creating a Dynamic List Element explains how to code dynamic list elements.

4. The marketing team uses the Engage recommendation wizard to create and then configure the recommendations. They assign the appropriate template to the appropriate recommendation.

5. Using the Engage product and content asset forms, the marketers rate how important the assets are to each segment, and, therefore, to the individual visitors who become members of those segments. (Typically, you assign ratings to flex parents, such as product parents, instead of to individual assets.)

6. For each related items recommendation, the marketers configure the relationships maintained by those recommendations by assigning related assets in the flex asset or flex parent forms. (Typically, relationships are configured among flex parents, such as product parents, instead of individual assets.)

48.3 About Creating a Dynamic List Element

For websites that use dynamic list recommendations, the dynamic list elements return the lists of recommended assets. A dynamic list element is an instance of the `CSElement` asset type. This asset type enables the transfer of the dynamic list element to the delivery system at the time of publishing.

A dynamic list element must return a list named `AssetList`. The set of assets that becomes your `AssetList` must have the following traits:

- It must contain only assets of the types that you want to recommend.

- It must contain the IDs of the assets that you want to recommend.

- It should contain the assets' confidence ratings, although this is optional.

The following sample code is an excerpt from a dynamic list element. Line 2 in the following code adds a constraint to the `ssprod` searchstate, filtering it to find items with a browse category of `Fund Type`. Line 3 adds another constraint to the `ssprod`

searchstate, creating an assetset composed entirely of `Product` assets. Finally, line 4 turns the assetset created in line 3 into the `AssetList` list.

```
<SEARCHSTATE.CREATE NAME="ssprod"/>
<SEARCHSTATE.ADDSIMPLESTANDARDCONSTRAINT NAME="ssprod" TYPENAME="PAttributes"
ATTRIBUTE="BrowseCategory" VALUE="Fund Type"/>
<ASSETSET.SETSEARCHEDASSETS NAME="asprod" CONSTRAINT="ssprod" ASSETTYPES="Products"/>
<ASSETSET.GETASSETLIST NAME="asprod" LISTVARNAME="AssetList"/>
```

When you have completed coding your dynamic list elements, provide their names and information about what sort of content they return to the users who create the recommendation assets.

49

Working with Memory-Centric Visitor Tracking

Site visitors who browse Oracle WebCenter Sites: Engage assets typically provide information about themselves in a personal profile. Demographics information is also collected as the visitors browse. To prevent overloading the WebCenter Sites database with large amounts of visitor data, and therefore, to improve performance, memory-centric visitor tracking was developed for Engage assets.

For information about the memory-centric method for tracking visitors and requirements for implementing this method, see these topics:

- About Memory-Centric Visitor Tracking

- Enabling Memory-Centric Visitor Tracking

- How Memory-Centric Visitor Tracking Works

49.1 About Memory-Centric Visitor Tracking

When large numbers of visitors browse a website, storing all of their personal information in a single repository degrades the performance of the delivery system. For this reason, websites that collect demographic information about visitors often require an additional repository to help improve site performance. You can gain additional performance by preventing load on the WebCenter Sites database.

WebCenter Sites supports add-on repositories, enabling Engage developers to implement a repository of their own choice to store visitor scalar attribute values. Custom code must be written to store and retrieve visitor information to and from the repository. In addition, the WebCenter Sites memory-centric visitor tracking method must be enabled. Differences between memory- and database-centric methods are outlined in this overview. Information about enabling memory-centric tracking is provided in the rest of this chapter, followed by diagrams illustrating how memory-centric visitor tracking works.

This section includes the following topics:

- Database-Centric Model

- Memory-Centric Model

49.1.1 Database-Centric Model

In database-centric visitor tracking, performance issues on Engage-enabled websites arise for the following reasons:

- All visitor information is stored in the WebCenter Sites database. Multiple database accesses are required to store and retrieve information.

- When dynamic recommendations are used and a large amount of data is returned, WebCenter Sites logs a greater number of dependencies in cache, which leads to performance degradation. To restore performance, the `render:overridedeps` tag limits the number of dependencies that are logged in the WebCenter Sites database. This tag applies to both the database and memory-centric models.

49.1.2 Memory-Centric Model

Memory-centric tracking improves system performance by reducing load on the WebCenter Sites database.

- All computations are performed in memory:

  - When custom code retrieves visitor scalar attribute values (such as age and gender) from the add-on repository, the memory-centric model stores the retrieved values in memory and uses them with history attribute values to compute segments to which the visitor belongs.

  - Memory-centric tracking computes statistics on various history attribute values in memory and then caches the statistics. Statistics include sums, counts, oldest, and newest.

- Alias and history attribute values are stored in the WebCenter Sites database. It helps reduces both the volume of visitor information and the number of calls required to access the information.

As in database-centric tracking, developers can use the `render:overridedeps` tag to specify the number and types of Engage asset dependencies to log in the WebCenter Sites database. Enabling memory-centric tracking requires setting a property in `wcs_properties.json` and writing supporting template code.

49.2 Enabling Memory-Centric Visitor Tracking

You can enable memory-centric visitor tracking by manually adding the `vis.useSessionVisitorConnection` property to the `wcs_properties.json` file on the delivery system. The code that you write should support memory-centric visitor tracking. You can also batch save History attributes to reduce memory usage.

See these topics:

- Visitor Tracking Property

- Supporting Code

- Batch-Saving History Attributes to the Database

49.2.1 Visitor Tracking Property

Memory-centric visitor tracking is enabled by the property `vis.useSessionVisitorConnection`, which must be manually added (using the Property Management Tool) to the WebCenter Sites `wcs_properties.json` file on the delivery system:

- Setting `vis.useSessionVisitorConnection=true` enables memory-centric visitor tracking. Supporting template code must also exist on the delivery system for Engage visitors to be correctly tracked; pre-existing code can be reused. See Supporting Code.

- Setting `vis.useSessionVisitorConnection=false`, leaving it blank, or omitting it from `wcs_properties.json` enables the database-centric method.

> **Note:**
>
> Visitor attributes are not stored in the database when using the memory mode. Therefore, on all content management systems, `vis.useSessionVisitorConnection` must be either set to `false` or omitted, thus enabling the database-centric method, which allows visitor attributes to be created and otherwise managed. (Visitor attribute management is supported only on content management systems that are enabled for database-centric tracking. It is not supported on delivery systems. Visitor attributes must be published to the delivery system.)

49.2.2 Supporting Code

The skeleton template in this section shows the type of code that must be written to support memory-centric tracking.

> **Note:**
>
> How the template code executes depends on the value of the `vis.useSessionVisitorConnection` property. If the value is set to `false`, the database method of tracking visitors is used. If the value is set to `true`, memory-centric tracking takes effect.

```
vdm:setalias
retrieve visitor scalar attributes from add-on repository
vdm:setscalar
commercecontext:calculatesegments
commercecontext:getrecommendations
render Engage assets
vdm:recordhistory
...
render:overridedeps
```

See How Memory-Centric Visitor Tracking Works.

49.2.3 Batch-Saving History Attributes to the Database

As of WebCenter Sites 7.5 Patch 2, history attributes are saved first to memory, then to the file system, and finally as a batch to the WebCenter Sites database. Batch-saving prevents excessive memory usage (which would otherwise occur when data is produced faster than it can be saved to the database). The batch-save process batches one `HistoryAttributeDef` table at a time from the file and saves the tables to the database in bulk.

Batch saving to the database requires Engage to have access to database information. Database access is enabled by adding a file named `<dataSourceName>.properties` to the classpath to specify the following information: `driver`, `url`, `user`, and `password`. For example, if the data source name is `csDataSource`, then a file named `csDataSource.properties` must be placed in the classpath. In our example, properties in the file are set as follows:

```
driver=com.jnetdirect.jsql.JSQLDriver
url=jdbc:JSQLConnect://localhost:1433/database=TomcatDB
user=tomcatuser
password=tomcatuser
```

The following Java JVM parameter controls the time interval at which threads are spawned for asynchronous batch-saving of history attributes from memory to the WebCenter Sites database through the file system:

```
-Dvisitor.SyncInterval=<seconds>
```

If left unspecified, the parameter value defaults to 30 seconds.

49.3 How Memory-Centric Visitor Tracking Works

A first-time visitor creates a personal profile, a tag in the template code logs the visitor's alias to the WebCenter Sites database, memory-centric visitor tracking logs a unique ID per visitor, the existing database-centric method logs a unique ID per visit, and the visitor's personal information such as age and gender are stored to the add-on repository. WebCenter Sites computes the visitor's segments from scaler and History attribute values. It computes the sums and counts and stores the results in memory. WebCenter Sites compares the visitor's scalar values in memory to those in the segment rules, the history attribute values to those in the segment rules, the sums and counts in memory to the sums or counts in the segment rules. From the comparisons, WebCenter Sites determines the visitor's segments. Then it determines which Engage assets are of most interest to the visitor.

Diagrams in these topics show how memory-centric tracking works.

> **Note:**
>
> In the diagrams that follow, the add-on repository is a system of your own choice, used to store and retrieve visitor scalar attribute values. Custom code must be written to store and retrieve the values.

See these topics:

- Visitor Detection
- Retrieval of Scalar Values
- Collection of History Attribute Values
- Computation of Sums and Counts
- Computation of Segments
- Display of Recommended Assets
- Logging of Dependencies

49.3.1 Visitor Detection

A first-time visitor enters the site and creates a personal profile. The following events occur:

1. The `vdm:setalias` tag in the template code is used to log the visitor's alias to the WebCenter Sites database. In subsequent sessions, the visitor is automatically

recognized. As a result, a single ID unique to the visitor is kept in the database, which improves performance.

Memory-centric visitor tracking logs a *unique ID per visitor*, whereas the existing database-centric method logs a *unique ID per visit* even for returning visitors. Extra IDs create extra load on the database and reduce performance.

2. The visitor's scalar values (personal information, such as age and gender) are stored to the add-on repository (by custom code).

In the remaining steps, WebCenter Sites gathers and computes information that helps determine the visitor's segments and Engage assets to display to the visitor.

This figure shows the flow for visitor detection:

Figure 49-1   Visitor Detection Flow
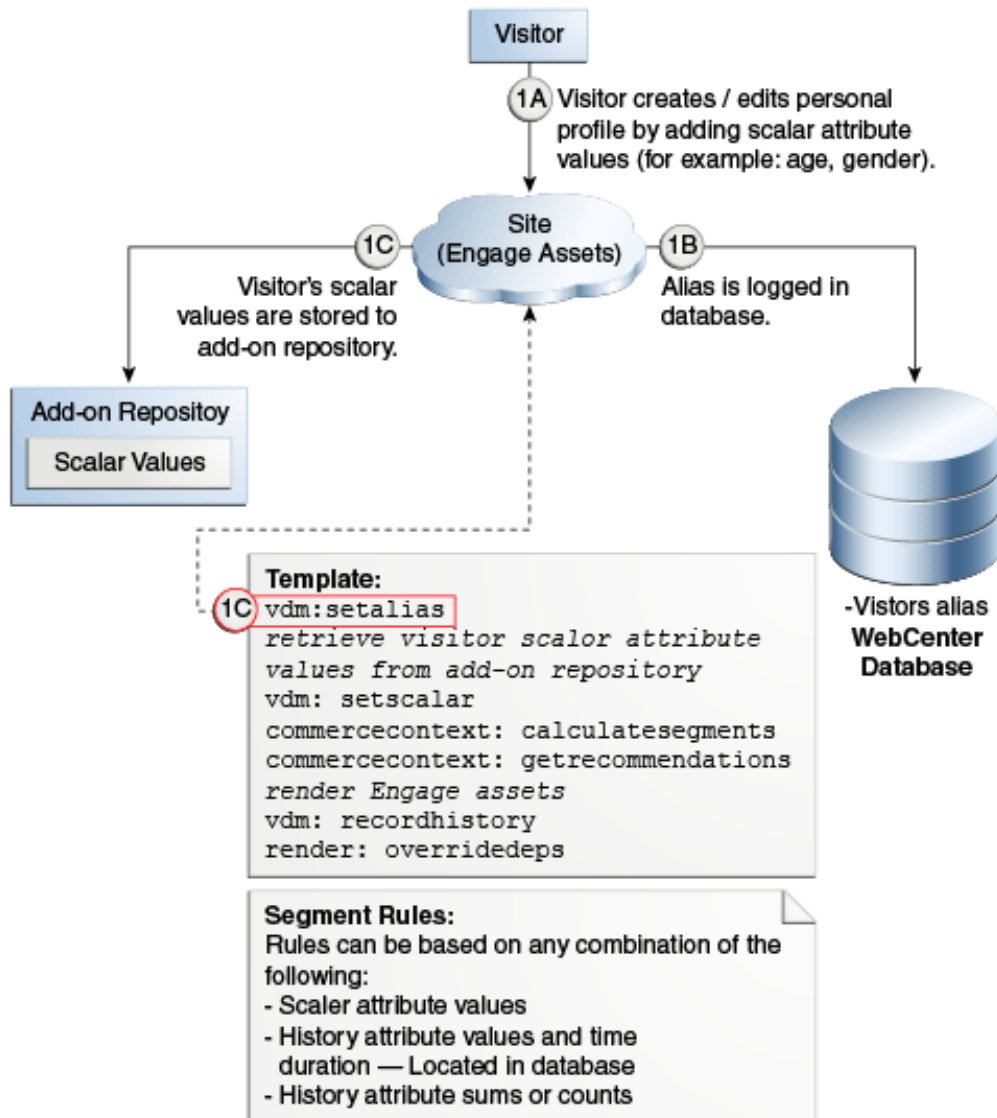
49.3.2 Retrieval of Scalar Values

WebCenter Sites begins computing the visitor's segments, starting with scalar values:

1. WebCenter Sites retrieves scalar values from the add-on repository (using custom code).

2. WebCenter Sites sets retrieved scalar values in memory, using the `vdm:setscalar` tag.

    The session-based implementation stores visitor information (scalar values) only in the http session. Therefore, when a new session begins, visitor information is made available through the add-on repository, instead of the WebCenter Sites database. A new level of caching has been added to optimize the performance of querying for visitor scalar values.

    This figure shows the flow for scalar value retrieval:

    Figure 49-2   Scalar Value Retrieval Flow
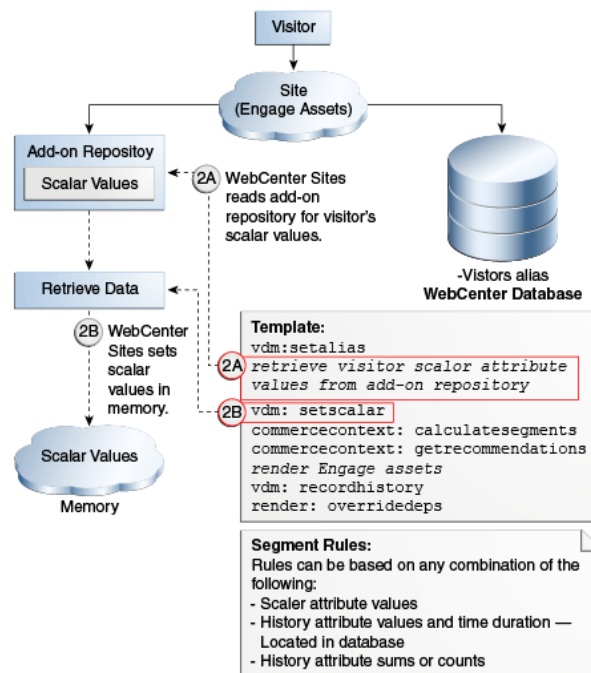
49.3.3 Collection of History Attribute Values

History attribute values are also required for computing a visitor's segments. During the session, history attribute values are collected cyclically:

1. The `vdm:recordhistory` tag is used to collect history attribute values into memory at one-minute intervals.

2. The history attribute values are saved to the file system and then written to the WebCenter Sites database in batches. The memory is cleared and the cycle starts again.

    Visitor history attribute values are collected first in memory to avoid the need for acquiring database connections at each call. They are saved to the file system to enable batch saving to the database, which minimizes memory and database connections usage, and increases the availability of history attribute values.

When history attribute values are committed to the database, computation of sums and counts begins, as shown in Computation of Sums and Counts.

The following figure shows the flow for history attribute values:

Figure 49-3   History Attribute Value Flow
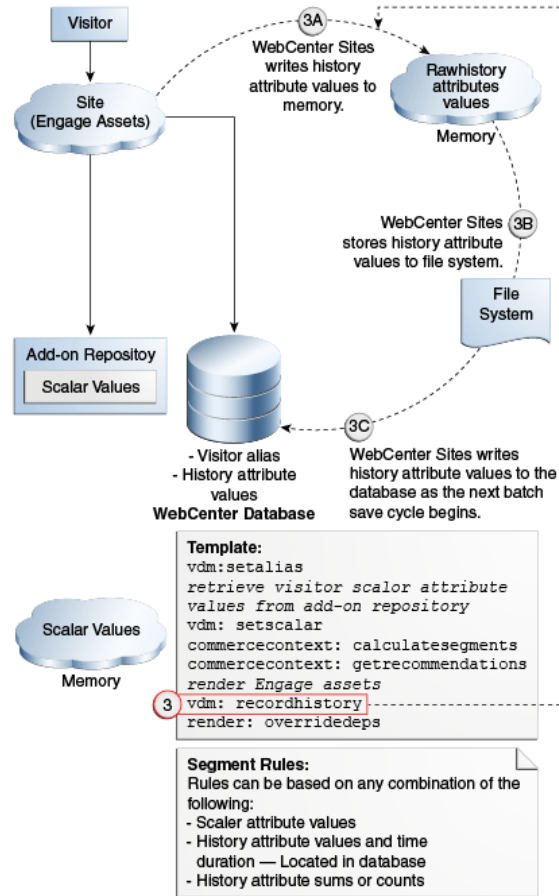
49.3.4 Computation of Sums and Counts

After the first set of history attribute values is collected and written to the database, WebCenter Sites reads the database at 1-minute intervals, computes the sums and counts, and stores the results in memory.

Caching of sums and counts significantly reduces database queries, while the 1-minute interval makes the computation of sums and counts accurate to within 1 minute.

The following figure shows the flow for computing sums and counts:

Figure 49-4   Computing Sums and Counts Flow
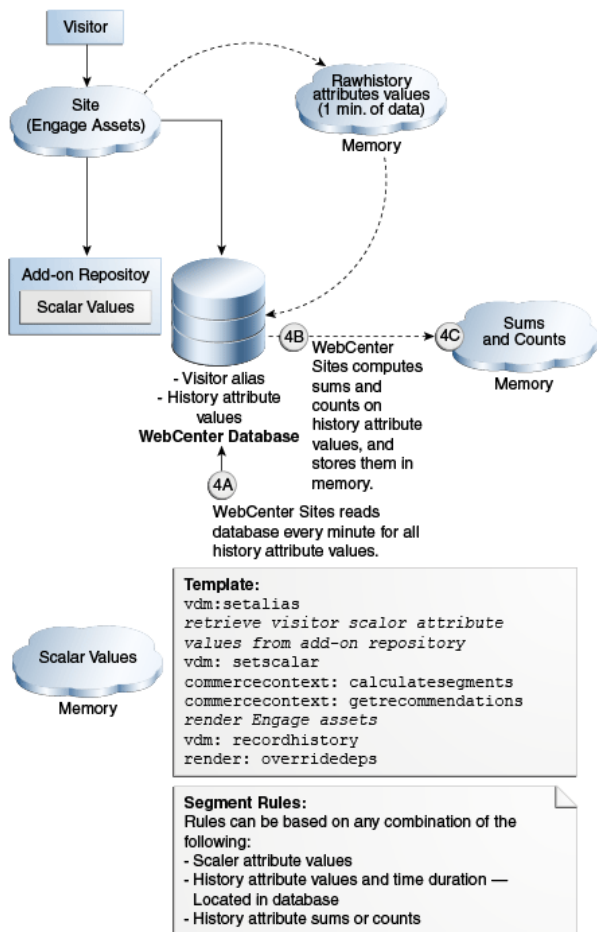
49.3.5 Computation of Segments

At this point, WebCenter Sites has enough information to compute the visitor's segments. Using the `commercecontext:calculatesegments` tag, WebCenter Sites does the following:

1. Compares the visitor's scalar values in memory to those in the segment rules.

2. Compares the history attribute values in the database to those in the segment rules.

3. Compares the sums and counts in memory to the sums or counts in the segment rules.

From the comparisons, WebCenter Sites determines the visitor's segments.

This figure shows the flow for computing segments:

Figure 49-5   Computing Segments Flow
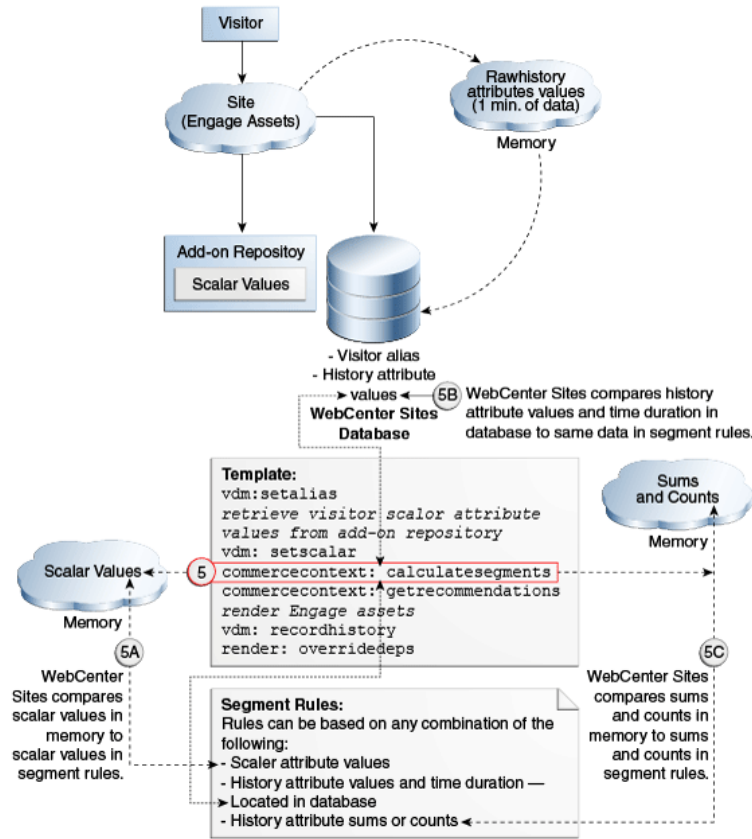
49.3.6 Display of Recommended Assets

Having determined the visitor's segments, WebCenter Sites uses the
`commercecontext:getrecommendations` tag to determine which Engage assets
are of most interest to the visitor. Templates for Engage assets display the
recommended assets to the visitor.

Further performance gains are achieved by use of the `render:overridedeps` tag, as
explained in Logging of Dependencies.

The following figure shows the flow to display recommended assets:

Figure 49-6   Displaying Recommended Assets Flow
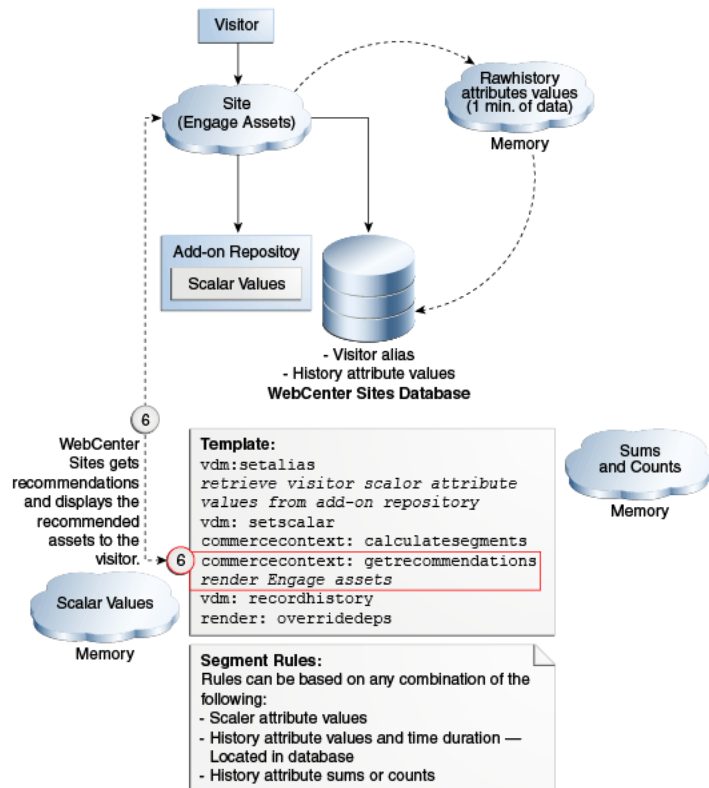
49.3.7 Logging of Dependencies

The `render:overridedeps` tag is used to reduce the number of dependencies logged in cache. The tag must be inserted in the template (or CSElement), just before the `</cs:ftcs>` tag, to remove all the existing dependencies logged on the page and to log the dependencies that are specified by the `render:overridedeps` tag. The possible dependencies are:

- Unknown dependencies

- Dependencies on a single specific asset

- Unknown dependencies for a specific asset type

The `render:overridedeps` tag takes the following parameters:

- `cid`: ID of the asset of type `c`. When combined with `c`, it works like asset, a dependency is logged against the asset determined by `c` and `cid`.

- `c`: Type of asset.

- `deptype`: Dependency type.

Which parameters are used and how they are set determines the dependency that is logged:

- When `deptype` alone is specified and set to `unknowndeps`, `render:overridedeps` logs an unknown dependency.

- When `c` and `cid` are both specified, `render:overridedeps` logs a dependency on the asset specified by `c` and `cid`.

- When `c` and `deptype='unknowndeps'` are both specified, `render:overridedeps` logs an unknown dependency on the asset type specified by `c`.

> **Note:**
>
> Use `render:overridedeps` carefully. The only way to flush page caches is to edit the asset for which dependencies are logged. For example, when logging dependencies for a single asset (or asset type), flush the cache by editing the asset for which dependencies are logged. Users must know which asset(s) to edit.

50

Coding Engage Pages

With Oracle WebCenter Sites: Engage you can design online sites that gather information about visitors which is useful in discerning products and services that visitors would be interested in. Based on this information, these sites display personalized promotional messages for each visitor.

Topics:

- Commerce Context and Visitor Context
- Identification of Visitors and Linking Sessions
- Collection of Visitor Data
- Coding of Site Pages That Collect Visitor Data
- Templates and Recommendations
- What You May Need to Know About Shopping Carts and Engage
- Debugging Site Pages

> **Note:**
>
> This chapter refers to specific XML tags that you use to accomplish the tasks being described. In all cases, there are also equivalent JSP tags. The XML and JSP tags are all documented in the *Oracle Fusion Middleware Tag Reference for Oracle WebCenter Sites Reference*.

50.1 Commerce Context and Visitor Context

At an Engage site, a visitor context is created during a visitor's session. This context includes session objects such as the shopping cart, visitor segments and promotions. The commerce context encompasses the visitor context and gives you access to it.

Five types of session objects are placed in the visitor context:

- Current shopping cart.
- List of segments that the visitor belongs to.
- List of promotions that the visitor qualifies for.
- Time object that is used for calculating time-based rules for segments and promotions.
- Utility object that gives you, the developer, access to product attributes.

There are two sets of XML and JSP object methods that serve as your interface to these contexts:

- Commerce context methods, which you use to place objects in the visitor context.

- Visitor Data Manager methods, which you use to gather, store, and retrieve visitor data and to associate a visitor's data with the correct visitor.

50.2 Identification of Visitors and Linking Sessions

Engage creates a unique visitor ID for each visitor for each session. It stores these IDs in the VMVISITOR table in the WebCenter Sites database. The data gathered for a visitor during that session is identified by that visitor ID. To link the data gathered from one session to the data from another, your site pages must assign aliases that link those visitor IDs.

You use the following Visitor Data Manager object method to create an alias:

```
<VDM.SETALIAS KEY="keyvalue" VALUE="aliasvalue"/>
```

When you use this tag, Engage associates the visitor session ID with the alias, and writes them both to the VMVISITORALIAS table.

Figure 50-1   VMVISITORALIAS Table

The values in this table link the data that is gathered in separate sessions to the same visitor because the alias provides a link to the visitor IDs that are recorded for that visitor. In the illustration above, the data recorded in the session associated with the visitor ID 973717492772 is linked to the data associated with the visitor ID 973717564355 because they have aliases with the same key/value pair.

All visitor information is associated with sessions that are linked through common aliases. That is, aliases with the same key/value pairs can be accessed during the current session. It is considered current visitor information. You can create aliases with cookies, with login IDs, or with any other unique identifier that your site uses to recognize visitors.

The VMVISITORALIAS table grows quickly. See Visitor Tables (Engage) in *Oracle Fusion Middleware Administering Oracle WebCenter Sites*.

50.3 Collection of Visitor Data

You need to program your online pages for collecting, validating, and writing the visitor data to the WebCenter Sites database. There are three Visitor Data Manager object methods that write visitor information to the database.

- `<VDM.SETSCALAR ATTRIBUTE="attribute" VALUE="value"/>` records visitor attributes.

- `<VDM.RECORDHISTORY ATTRIBUTE="attribute" LIST="valuelist"/>` records history definitions.

- `<VDM.SAVESCALAROBJECT ATTRIBUTE="attribute" OBJECT="objectname"/>` records visitor attributes of type binary. The demo site delivered with Engage uses this method to store shopping carts across sessions and to store saved searches for visitors.

> **Note:**
>
> Because these tags write information to the database, they can be a factor in the performance of your delivery system. Be sure to use them efficiently.

This table shows the tables that store the visitor data:

Table 50-1   Tables to Store Visitor Data

XML or JSP Object Method	Database Table That It Writes To
VDM.SETSCALAR vdm:setscalar	VMVISITORSCALARVALUE
VDM.SAVESCALAROBJECT vdm:savescalarobject	VMVISITORSCALARBLOB
VDM.RECORDHISTORY vdm:recordhistory	VMz------------ (These tables are dynamically generated for each history definition. Engage creates a unique table for each one.)

These tables grow quickly. See Visitor Tables (Engage) in *Oracle Fusion Middleware Administering Oracle WebCenter Sites*.

There are also several Visitor Data Manager object methods that retrieve this information from the WebCenter Sites database. See Logging and Debugging Errors.

For information about these and other Engage XML and JSP tags, see the *Oracle Fusion Middleware Tag Reference for Oracle WebCenter Sites Reference*.

50.4 Coding of Site Pages That Collect Visitor Data

It takes you three steps to code site pages that collect visitor data. Create forms to capture the data, create a submit page to validate the data, and program the submit page to write the validated data to the database.

The general steps to code your site pages to collect visitor data are:

1. Create forms to capture the data that you need your visitors to manually provide. It is a good practice to create form fields with names that match the names of the attributes that you created. See Creating Visitor Data Assets.

   Attributes are listed by their descriptions rather than by their names in the Engage Segment forms. Be sure that you do not confuse their attribute names with attribute descriptions when you are creating form fields or writing values to the WebCenter Sites database.

2. Create a submit page that validates the data that the visitor entered in the fields (either by using JavaScript or with a server-side validation method). The input data must comply with the constraints that you set for the attributes. For example, when a visitor attribute of type string has a length of 30, the form must not try to submit data from the form field with a length of 31.

3. Program the submit page to write the validated data to the WebCenter Sites database. Be sure to use the names of the attributes and history definitions and not their descriptions. Here are some examples:

See these topics:

- Example 1: Visitor Attributes

- Example 2: History Definition

- Example 3: Visitor Attribute of Type Binary

50.4.1 Example 1: Visitor Attributes

The following example uses attribute names to write the registration information to the database:

```
<!-- Write the registration information to the database.-->
<VDM.SETSCALAR ATTRIBUTE="name" VALUE="Variables.name"/>
<VDM.SETSCALAR ATTRIBUTE="age" VALUE="Variables.age"/>
<VDM.SETSCALAR ATTRIBUTE="jobdesc" VALUE="Variables.jobdesc"/>
```

50.4.2 Example 2: History Definition

Because history definitions hold multiple values as an aggregate, you must create a list of the data before you can write it to the database. In this example, a form writes an order to the WebCenter Sites database:

```
<!-- Write the order details to a list. -->
<!-- assume that Variables.order_id is set to the order id -->
<!-- assume that Variables.wasCouponUsed is set to 1 (yes) or 0 (no) -->
<!-- assume that Variables.shippingtype is set to UPS or FedEx -->
<!-- assume that Variables.order_price is set to the total amount of the
order -->

<LISTOBJECT.CREATE NAME="histList" COLUMNS="orderid, shippingtype, price,
couponUsed"/>
<LISTOBJECT.ADDROW NAME="histList" orderid="Variables.order_id"
shippingtype="Variables.shippingtype" price="Variables.order_price"
couponUsed="Variables.wasCouponUsed"/>
<LISTOBJECT.TOLIST NAME="histList" LISTVARNAME="itemList"/>

<!-- Write the list to the history definition named visitorOrderHistory in the
WebCenter
Sites database.-->
<VDM.RECORDHISTORY ATTRIBUTE="visitorOrderHistory" LIST="itemList"/>
```

You can use that record to determine information about how many orders a visitor had made, when their first or last purchase was, and the total amount they've spent.

50.4.3 Example 3: Visitor Attribute of Type Binary

Binary visitor attributes allow you to convert an object from the WebCenter Sites name space into a binary form. The following procedure uses two visitor attributes of type binary: one to store shopping carts across sessions and one to store saved searches:

1. To gather data about visitor behavior (such as clickstream information), program your pages to collect the data without using input forms.

    For example, you can use a history definition to record the number of times a visitor browses the site.

2. Whenever visitor data is written to the database, segments and promotions can also change. After any change to visitor data, be sure to recalculate the segments and promotions lists. There are two Commerce Context object methods that you can use:

    - COMMERCECONTEXT.CALCULATEPROMOTIONS

    - COMMERCECONTEXT.CALCULATESEGMENTS

    COMMERCECONTEXT.CALCULATEPROMOTIONS recalculates both the segments that the visitor belongs to and the promotions that apply to those segments.

3. Whenever visitor data is written to the database, ratings for assets can also change. After any change to visitor data, be sure to refresh the ratings of any assets that are in an existing asset set.

    Use the ASSETSET.ESTABLISHRATINGS tag to refresh the asset ratings of the assets in a set.

    > **Note:**
    >
    > For information about these and other Engage XML and JSP object methods, see the *Oracle Fusion Middleware Tag Reference for Oracle WebCenter Sites Reference*.

50.5 Templates and Recommendations

You need to create a template element that invokes a recommendation asset and displays the items that the recommendation returns. You use the key Commerce Context object method to invoke a recommendation asset. This method retrieves and lists the assets that meet the recommendation criteria. The object method invokes other methods that calculate the segment and promotion lists.

```
<COMMERCECONTEXT.GETRECOMMENDATIONS COLLECTION="recommendationname"
[LIST="inputlist" VALUE="rating" MAXCOUNT="assetcount"] LISTVARNAME="assetlist"/>
```

This method retrieves and lists the assets that match the recommendation constraints passed to the method. It uses the following arguments:

- COLLECTION: The name of the recommendation. To use the same Template for several recommendations, code it to supply the recommendation identity through a variable.

- LIST: The name of the list of assets. This is the name that you want to be used as the input for the calculation.

You use this argument when the recommendation named by COLLECTION is a context-based recommendation. Columns are assettype and assetid. You can create this list by creating a list object and adding rows for each asset that you want to use as input.

- VALUE: The default rating for assets that do not have one. If you do not declare a value, unrated assets are assigned a default rating of 50 on a scale of 0-100. It is recommended that you keep this value set to 50.

- MAXCOUNT: (Optional.) The maximum number of assets to return. Use this value to constrain the list of recommended assets.

- LISTVARNAME: The name that you want to assign to the list of assets. Its columns are: assettype and assetid.

The object method invokes the methods that calculate the segment list and the promotion list, if they have not yet been created and placed in the visitor context. Remember that promotions do not have templates, they override the Template that a recommendation is using. Rather than the items identified by the recommendation asset, the object method returns the promotion asset's ID if promotions apply to the current visitor and override the recommendation named by the COLLECTION argument.

> **Note:**
>
> The COMMERCECONTEXT.GETSINGLERECOMMENDATION object method returns one recommended asset based on the recommendation criteria passed to the method. Typical uses for this method are to feature one product or to put one product on sale. For information about this object method and its JSP equivalent, see the *Oracle Fusion Middleware Tag Reference for Oracle WebCenter Sites Reference*.

50.5.1 Creating Templates for Recommendations

Before you begin coding the Templates for recommendations, be sure to complete the following tasks:

- Meet with the marketing team to define all the merchandising messages that you want to display on your site and plan how to represent those messages in recommendations and promotions. For example, do you want to display a list of links to other products? What information should the link include? The product name only or also the price? What will be displayed when a recommendation returns a promotion rather than a list of assets?

- Determine where and on which pages the recommended assets from each recommendation are displayed.

To use templates to render items that are returned by recommendation assets, you must complete at least the following basic steps:

1. Create a template element that invokes a recommendation asset. Use the object method described in the preceding section.

2. Code the template to display the items that are returned by the recommendation. The returned items are stored in a variable designated by the LISTVARNAME argument. This list includes the asset IDs and asset types of those items. Use that

information to extract the asset attributes that you want to display (for example, `Name`, `Price`, and `SKU`).

You can use the `ASSETSET.SETLISTEDASSETS` and `ASSETSET.GETASSETLIST` object methods to sort and display the returned assets and their attributes.

3. Open Engage. Under **New**, choose **Template**. Create a corresponding `Template` asset for this `Template` element. Enter a name that describes what the element does so that when you create a recommendation asset, you know which `Template` to assign to it. Identify the path to the element (its location in the element catalog) in the **Element Name** field.

4. Publish the `Template` asset when other assets are published.

5. Render the recommendations on the appropriate site pages.

50.5.2 Creation of Templates for Recommendations Using Oracle Real-Time Decisions

Oracle Real-Time Decisions (RTD) is an engine that helps site visitors make decisions by recommending the best options when they make their choices. When the `commercecontext:getrecommendations` tag is invoked, it sends a list of recommendations and certain visitor profile information to Oracle RTD. Oracle RTD then ranks the recommendations and, using the `maxcount="<n>"` parameter, returns a refined list of *n* recommendations best suited for the visitor's profile.

> **Note:**
>
> This release of WebCenter Sites supports Oracle Real-Time Decisions (RTD) version 3.0 series. RTD 11*g* is not supported.

Oracle RTD (and other engines) can be integrated with WebCenter Sites by use of the following tags:

- The `commercecontext:getrecommendations` tag, where `engine` and `engineparameters` are generic parameters that can be set to invoke any engine. For integration with Oracle RTD, the engine parameter must be set to `rtd` and `engineparameters` must name a list of the following Oracle RTD-specific parameters: `advisor` (integration point), `attributes` (visitor profile), `sessionkey`, and `assetattributes`.

- The `commercecontext:inform` tag, which also takes the parameters `engine` and `engineparameters`. For this tag, `engineparameters` must name a list of the following parameters: `informant` (integration point), `attributes` (visitor profile), and `sessionkey`.

> **Note:**
>
> The `choices` parameter (choice of recommendations) was removed from this tag. The list of choices is now passed to RTD with the list `tag` attribute.

After Oracle RTD learns the list of attributes from the `commercecontext:getrecommendations` tag, you, the developer, can use the `sessionkey` parameter to map visitors to their profiles for subsequent

commercecontext:inform calls. For example, after a
commercecontext:getrecommendations call has been made to Oracle RTD, the
next call from the same visitor may pass an empty string for attributes and use the
same sessionkey to get recommendations.

> **Note:**
>
> Using the commercecontext:getrecommendations and
> commercecontext:inform tags to invoke Oracle RTD requires adding the
> following properties to the wcs_properties.json file:
> rtd.inline.service.name and rtd.host.

For more information about the commercecontext tags, parameter definitions,
integration with Oracle RDT, and sample code, see the *Oracle Fusion Middleware Tag
Reference for Oracle WebCenter Sites Reference*. For information about the
rtd.inline.service.name property, rtd.host property, and
rtd.choiceId.pattern property, see the *Oracle Fusion Middleware Property Files
Reference for Oracle WebCenter Sites*.

50.6 What You May Need to Know About Shopping Carts and Engage

When you code your shopping cart pages using the shopping cart interface with
Engage there are some facts and tips that you should keep in mind.

- If your site uses promotions, you must code your cart pages to apply the
  discounts from the promotions.

  Use the COMMERCECONTEXT.DISCOUNTCART and
  COMMERCECONTEXT.DISCOUNTTEMPCART object methods to apply promotional
  discounts to the shopping cart.

- It is a good practice to clear existing discounts from the cart before applying them
  again.

- You can store carts across sessions by writing them to the database as a visitor
  attribute of type binary (a scalar object). Be sure to write the cart object to the
  database each time the cart is modified.

- If your site uses a visitor login feature, there can be conditions under which you
  should merge shopping carts. For example, a visitor adds products to her cart
  before she logs in. Then, when she logs in, Engage finds a stored cart that also has
  items in it. In such a case, merge the carts.

For information about the CART object methods and their JSP equivalents, see the
Oracle Fusion Middleware Tag Reference for Oracle WebCenter Sites Reference.

50.7 Debugging Site Pages

During development phase, you must verify that session linking is set up correctly,
specific attributes obtain the value that you expect, and recommendations return the
items that you expect. There are several Engage object methods that you can use to
retrieve and review information and values by writing information to a browser
window or to the JRE log.

This topic lists the Visitor Data Manager object methods that you probably use the
most. For information about these and any other XML and JSP object methods, see the
Oracle Fusion Middleware Tag Reference for Oracle WebCenter Sites Reference.

See these topics:

- Session Links

- Visitor Data Collection

- Recommendations and Promotions

50.7.1 Session Links

Use the following Visitor Data Manager object methods to verify that pages that handle session linking are creating the aliases correctly:

- `<VDM.GETALIAS KEY="keyvalue" VARNAME="varname"/>`

  Retrieves an alias.

- `<VDM.GETCOMMERCEID VARNAME="varname"/>`

  Retrieves the visitor's commerce ID from session data.

- `<VDM.GETACCESSID KEY="pluginname" VARNAME="varname"/>`

  Retrieves the visitor's access ID from session data.

50.7.2 Visitor Data Collection

Use the following Visitor Data Manager object methods to retrieve values stored for specific visitor attributes, history attributes, and history definitions (records):

- `<VDM.GETSCALAR ATTRIBUTE="attribute" VARNAME="varname"/>`

  Retrieves a specific visitor attribute.

- `<VDM.LOADSCALAROBJECT ATTRIBUTE= "attribute" VARNAME= "varname"/>`

  Retrieves (materializes) an object stored as a visitor attribute of type binary.

- `<VDM.GETHISTORYCOUNT ATTRIBUTE="attribute"VARNAME="varname"[STARTDATE="date1" ENDDATE="date2"LIST="constraints"]/>`

  Retrieves the number of history definition records that were recorded for the visitor that match the specified criteria.

- `<VDM.GETHISTORYSUM ATTRIBUTE="attribute" VARNAME="varname"[STARTDATE="date1" ENDDATE="date2" LIST="constraints"]FIELD="fieldname"/>`

  Sums the entries in a specific field for the specified history definition.

- `<VDM.GETHISTORYEARLIEST VARNAME="varname" [STARTDATE="date1"ENDDATE="date2" LIST="constraints"]/>`

  Retrieves the timestamp of the first time the specified history definition was recorded for this visitor.

- `<VDM.GETHISTORYLATEST VARNAME="varname"[STARTDATE="date1"ENDDATE="date2" LIST="constraints"] />`

Retrieves the timestamp of the last time (that is, the most recent time) the specified history definition was recorded for this visitor.

50.7.3 Recommendations and Promotions

Use the following Commerce Context object methods to verify pages that display recommendations and promotions:

- `<COMMERCECONTEXT.CALCULATESEGMENTS/>`

  Lists the segments that the visitor belongs to. It examines the available visitor data, compares it to the data types that define the segments, and then lists the segments that are a match.

- `<COMMERCECONTEXT.GETPROMOTIONS LISTVARNAME="promotionlist"/>`

  Creates the list of promotions that the current visitor is eligible for.

- `<COMMERCECONTEXT.GETRATINGS ASSETS="assetlist" LISTVARNAME="ratinglist" DEFAULTRATING="defaultrating"/>`

  Calculates the ratings of the assets in a named list according to how important the asset is to this visitor based on the segments that the visitor belongs to.

- `<COMMERCECONTEXT.GETSEGMENTS LISTVARNAME="segmentlist"/>`

  Retrieves the list of segments that the current visitor belongs to.

Part XI

Running A/B Testing

Learn about your role in running Oracle WebCenter Sites: A/B Testing and how you can develop conversion tracking for measuring ROI.

Topic:

- Developing for A/B Testing

51

Developing for A/B Testing

With the Oracle WebCenter Sites: A/B Testing feature, marketers can test variants of website pages (both design and content variations) to determine which page variants produce best results.

For information on how marketers create and run A/B tests, see Working With A/B Testing in *Oracle Fusion Middleware Using Oracle WebCenter Sites*.

Topics:

- A/B Testing Prerequisites

- Scripting Templates for A/B Testing

- Updating Cache Criteria for A/B Testing

- Viewing A/B Test Details as JSON

- Understanding Confidence Algorithms

51.1 A/B Testing Prerequisites

What you need to do to be able to use the A/B Testing feature is, include the A/B Testing code element in the templates, enable the A/B Testing property in the `wcs_properties.json` file, and obtain Google Analytics IDs.

> **Note:** A/B test functionality for this Oracle WebCenter Sites product is provided through an integration with Google Analytics. To use the A/B test functionality for this Oracle product, you must first register with Google for a Google Analytics account that will allow you to measure the performance of an "A/B" test. In addition, you acknowledge that as part of an "A/B" test, anonymous end user information (that is, information of end users who access your website for the test) will be delivered to and may be used by Google under Google's terms that govern that information.

Before the A/B Testing feature can be used, a developer must do the following:

1. Include the A/B code element in templates on which A/B tests will be run. See Scripting Templates for A/B Testing.

2. Set the property `abtest.delivery.enabled` to true on the delivery instance, that is any instance that will deliver the A/B test variants to site visitors. The property should not be set on instances that are used only to create the A/B tests, because this will give false results. The `abtest.delivery.enabled` property is in the ABTest category of the `wcs_properties.json` file.

See A/B Test Properties in the *Oracle Fusion Middleware Property Files Reference for Oracle WebCenter Sites*.

3. Obtain all Google Analytics IDs and add them to the configuration.

Other actions must be taken by users and administrators as prerequisites for A/B testing. For a complete list, see Before You Begin A/B Testing in *Oracle Fusion Middleware Using Oracle WebCenter Sites*.

51.2 Scripting Templates for A/B Testing

You need to add a single line of code, a call element, to allow A/B Testing to operate. The element code you include will generate JavaScript code on the page. When the tested page loads in the browser, it will call back to the server to see if there are any A/B tests on this page.

This line must be added to templates on those pages to be used in A/B Testing. Many sites incorporate a design that includes a shared template on all pages. If your site uses this design, then it will only be necessary to add the call element to the single template. In the avisports sample website, a template named `Head` includes the call element.

To edit the template, follow these steps for each site that will use A/B Testing:

1. In the Admin interface, search for and open the template.

2. Click the **Edit** icon to open the template form for editing. Select **Element**.

The Element form within the template is displayed.

Figure 51-1   Element Fields for Head Template

3. In the **Element Logic** field, add the following line:

```
<render:callelement elementname="fatwire/includeABTesting"
args="c,cid,pagename"/>
```

It is recommended that you add this at the top of the code. This will ensure that this JavaScript is active before anything on the page is rendered.

> **Note:** The preceding sample code describes how to include the `includeABTesting` element using the `render:callelement` tag in a template. Alternatively, you can include the element using `Fragment API` in a `Controller`. See Developer's Samples Website.

4. Save the template.

From here, you need to add `ab` to the Cache Criteria for the page templates used on the site.

51.3 Updating Cache Criteria for A/B Testing

After you have added the call element code to templates to enable A/B Testing, you need to add `ab` to the Cache Criteria for the page templates used on the tested pages.

In the avisports sample website, the two page templates used are `SectionLayoutGreen` and `SectionLayoutOrange`. You will need to make this change on all page templates used on your site that are used in A/B Testing.

> **Note:** Creating a new template adds `ab` to the cache criteria by default.

To update the page template:

1. In the Admin interface, find and open the page template used on your website in A/B Testing.

2. Click the **Edit** icon to open the page template for editing. Select the **Site Entry** screen.

   The Site Entry form for the template displays.

Figure 51-2   Site Entry for A/B Testing Template

3. In the **Cache Criteria** field, enter ab to the beginning of the list of comma separated items. Do not delete any other entries.

4. Save the template.

   Repeat these steps for all page templates used on pages in A/B Testing.

51.4 Viewing A/B Test Details as JSON

Each A/B Testing asset, used through the WCS_ABTest asset type, contains a field that specifically lists the changes made between the original web page (the A page) and the test page (the B page). These changes are referred to as the differential data, and are stored in JSON format in the **Variations (JSON)** field.

To view the differential data:

1. In the Contributor interface, open the A/B Testing asset in Form mode.

2. Select the **Content** tab.

   The asset is displayed in Form mode.

3. The JSON used is displayed in the **Variations (JSON)** field. This describes all differences between the A and the B (and additional, if used) test pages.

If needed, the differential data stored in the JSON can be edited directly, although this is not recommended for contributors. Copy the JSON to a proper JSON editor, and make the changes necessary, then paste back in the **Variations (JSON)** field.

51.5 Understanding Confidence Algorithms

When marketers create A/B tests, they can select a confidence level for their results. This number determines confidence in the significance of the test results, specifically that conversion differences are caused by variant differences themselves rather than random visitor variations.

See Selecting the Confidence Level of an A/B Test in *Oracle Fusion Middleware Using Oracle WebCenter Sites*. This section provides details on how conversion confidence is calculated.

The conversion rate is the conversion event count divided by the view count. This is typically represented by p. The percent change of the conversion rate is determined by subtracting the p of page A from p of page B and then dividing by the p of page A. This is calculated per-user. For example, a user who is *converted* 1000000 times will only be counted once. The algorithm used to determine confidence is Z-Score.

The word "confidence" in this context is used to refer to a statistical computation used to determine how confident we are that the difference between the results for A and the results for B are actually caused by variations on the pages, and not just random variations between visitors.

The confidence interval is calculated using the Wald method for a binomial distribution.

Figure 51-3   Wald Method of Determining Confidence Intervals for Binomial Distributions

$$SE = \sqrt{\frac{p(1-p)}{sampleSize}}$$

The larger the sample, the greater the confidence in the results. A common threshold used in A/B Testing is a confidence interval with a 3% range from the final score. However, this is only a common use, and any plus/minus range can be used. The confidence interval is then determined for the conversion rate (p) by multiplying the standard error with that percentile range of standard normal distribution.

At this point the results must be determined to be significant; that is, that conversion rates are not different based on random variations. The Z-Score is calculated in this way:

Figure 51-4   Z-Score Calculation

$$Z\text{-}Score = \frac{p_{variationB} - p_{variationA}}{\sqrt{SE^2_{variationA} + SE^2_{variationB}}}$$

The Z-Score is the number of positive standard deviation values between the control and the test mean values. Using the previous standard confidence interval, a statistical significance of 95% is determined when the view event count is greater than 1000 and that the Z-Score probability is either greater than 95% or less than 5%.

Part XII

Customizing Blogs

You can customize the blog data model. Sample template code and guidelines for developing blog functionality on different CM sites is available for you.

Topic:

- Customizing Blog Components

52

Customizing Blog Components

You can customize the default blog components such as Blog flex family hierarchy, RSS feed URLs to render custom pages. You can also create blog pages and modify the code.

Topics:

- Customizing the Blog Asset Form
- Adding Blog Functionality to CM Sites
- Customizing URLs for the RSS Feed

52.1 Customizing the Blog Asset Form

You can modify the Blog flex family hierarchy as you need. When you want to display the attribute as a field in the blog asset form, all you need to do is, create a new blog attribute and then add that attribute to the blog asset definition.

> **Note:**
>
> To see the hierarchical relationships between blog categories and blog assets, create a tree tab for your own reference. For instructions about creating a tree tab, see Creating a Tree Tab in *Oracle Fusion Middleware Administering Oracle WebCenter Sites*.

This section includes the following topics:

- Creating a Blog Attribute
- Adding a Blog Attribute to the Blog Asset Definition

52.1.1 Creating a Blog Attribute

The blog attribute you create will be displayed as a field in the blog asset form once you add the attribute to the blog asset definition.

To create a blog attribute:

1. Log in to the Admin interface as a general administrator.

2. Select the site on which the Blogs component is enabled.

3. In the button bar, click **New**.

4. Click **New Blog Attribute**.

5. In the Blog Attribute form, fill in the fields.

Figure 52-1   Blog Attribute Form

Note:

The fields can differ significantly based on the data type that you select for your attribute.

- **Name**: Enter a name of up to 64 characters (the name cannot contain spaces).

- **Description**: Enter a short summary that describes the use or function of the attribute.

- **Value Type**: Select a data type for this attribute.

- **Asset Type**: If the attribute is of type asset, select an asset from the drop-down list.

- **Mirror Dependency Type**: If the attribute is of type asset, select a dependency type.

- **Folder**: (Optional) If the attribute is of type `blob`, enter a path to the directory in which you want to store the attribute values.

- **Allow Embedded Links**: If the attribute is of type `text`, `blob`, or `URL`, select whether links to other pages or websites can be embedded in the attribute's content field.

- **Number of Values**: Choose either **single** or **multiple** from the drop-down list, depending on the data type selected for the **Value Type** field.

- **Attribute Editor**: (Optional) To use an input type other than the default, in the **Attribute Editor** field, select the appropriate attribute editor for the field.

- (Optional) **Character Set**: To override the default ISO character set (ISO-8859-1), enter the character set you want to use for this attribute.

If you are creating a foreign attribute (keeping data in an external system) fill in the following fields:

- **Editing Style**: To make this attribute available to users in its native table on the external system, select **external**.

- **Storage Style**: Select **external**. See Creating Foreign Flex Attributes.

- **External ID**: Specify the name of the column that serves as the primary key for the table that holds this foreign attribute (the column that uniquely identifies the attribute).

- **External Table**: Enter the name of the table that stores this attribute.

- **External Column**: Enter the name of the column in the table specified in the **External Table** field that holds the value of the attribute.

6. Click **Save**.

   Now that you have created the attribute, add the attribute to the blog asset definition. For instructions, see the next section.

52.1.2 Adding a Blog Attribute to the Blog Asset Definition

To add an attribute to the blog asset form you must add the attribute to the blog asset definition.

To add an attribute to the blog asset definition:

1. Log in to the Admin interface as a general administrator, and select the site on which the Blogs component is enabled.

2. Access the blog asset definition's Inspect form:

   a. In the button bar, click **Search**.

   b. In the Search form, click **Find Blog Asset Definition**, and click **Search**.

   c. Select **BlogAssetDef**.

3. In the Inspect form, click **Edit**.

Figure 52-2   Blog Attribute Form

4. In the **Attributes** field, select the attribute(s) from the **Available** list and use the **Required** or **Optional** button to move the attribute(s) to the **Selected** list. Which button you choose determines whether the attribute(s) will be required or optional in the blog asset form.

5. Click **Save**.

   The attributes you selected are now included as fields in the blog asset form. When a user creates a blog asset, the new attributes will be displayed as either required or optional fields.

52.2 Adding Blog Functionality to CM Sites

Would you like to use the Blogs component on a different CM site? You need to first create blog pages for that site.

To add blog functionality to your website:

1. Create pages on the content management site that will be used to render blog assets on the website.

2. Copy the blog code from the default blog templates and CSElements to your site's templates and CSElements. How your site is set up determines the modifications you must make to the default code once you insert it into your templates and CSElements.

3. Add the `blogsperpage` parameter that is specified in the Blogs component's SiteEntry asset to your own site's SiteEntry asset.

See these topics:

- Creating Blog Pages

- Adding Blog Code

- Adding Blog Parameters to Your Site's SiteEntry Asset

52.2.1 Creating Blog Pages

Before creating blog pages, map out their types: the main blog page (which is the `FW_RecentBlogs` page in the Blogs component), category pages, and so on. Also determine your site's graphical, navigational, and functional features to create blog pages that will conform to the layout of your website.

To create blog pages:

1. Log in to WebCenter Sites as a general administrator, select the site on which you want to create the pages for displaying blogs to website visitors and then select the icon for the Oracle WebCenter Sites: Contributor interface.

2. In the menu bar, select **Content**, then select **New**, and then select **New Page**.

   A tab opens displaying the New Page form.

   Figure 52-3 New Page Form

3. In the New Page form, fill in the fields:

   - **Name**: Enter a name of up to 64 characters.

   - **Tags**: Enter a single word or phrase to attach to the page.

   - **Template**: Select the template that will render the page.

   - **Associated Items**: Add content (for example, related articles) to this field's **Drop Zone**.

4. Click the **Save** icon.

Now that you have created a page to display blog assets, code your site's templates and CSElements to call the new page and render blog functionality on your website.

52.2.2 Adding Blog Code

The Blogs component is configured to render sample blog pages. Your own site's layout is likely to differ from the layout of the sample blog pages. For example, your site may call a left navigation, while the sample blog pages call a right navigation. Instead of coding your templates from scratch to incorporate blog functionality, reuse the sample code by inserting it into your own templates and CSElements, then reconfiguring the code as necessary.

> **Note:**
>
> Your site's wrapper element renders the layout of your site. To ensure that your wrapper element renders blog pages in addition to existing pages, copy the relevant blog code from the default `FW_Wrapper` element to your site's wrapper element.

To add blog code to your site's layout template:

1. Log in to the Admin interface as a general administrator.

2. Select the site on which the Blogs component was installed.

3. Access the `FW_BlogLayout` template:

   a. In the button bar, click **Search**.

   b. In the **Search** list, select **Find Template**.

   c. In the **Search** field, enter `FW_BlogLayout`, and then click **Search**.

   d. Click **FW_BlogLayout**.

4. In the layout template's Inspect form, click **Edit**.

5. Copy the necessary code from the Blogs component's layout template and insert it into your own layout template:

   The following lines retrieve the site description from the wrapper and load the site:

```
<ics:if condition='<%=ics.GetVar("tid")!=null%>'>
  <ics:then>
    <render:logdep cid='<%=ics.GetVar("tid")%>' c="Template"/>
  </ics:then>
</ics:if>

<publication:load name='Publication' field="name"
    value='<%=ics.GetVar("site")%>'/>
<publication:get name='Publication' field="id" output="spubid"/>
<publication:get name='Publication' field="description" output="pubdesc"/>
```

   The following line retrieves the body of the page:

```
String sContainerTName = "FW_BlogContainer";
```

The following lines load the site, page, and asset descriptions:

```
 String sTitle = "";
if (!"Page".equals(ics.GetVar("c")))
{
   %><asset:load name='t2' type='<%=ics.GetVar("c")%>'
   objectid='<%=ics.GetVar("cid")%>' /><%
   %><asset:get name='t2' field='name' output='t2Name' /><%
   %><asset:get name='t2' field='description' output='t2Desc' /><%
   sTitle += ": "+(Utilities.goodString(ics.GetVar("t2Desc")) ?
   ics.GetVar("t2Desc") : ics.GetVar("t2Name"));
   } else if (Utilities.goodString( ics.GetVar("p") )) {
   %><asset:load name='t1' type='Page' objectid='<%=ics.GetVar("p")%>' /><%
   %><asset:get name='t1' field='name' output='t1Name' /><%
   %><asset:get name='t1' field='description' output='t1Desc' /><%
   sTitle += ": "+(Utilities.goodString(ics.GetVar("t1Desc")) ?
   ics.GetVar("t1Desc") : ics.GetVar("t1Name"));
}
```

The following line calls the style sheet for the blog layout template. The style sheet defines the look and feel of the sample blog pages. Because your site has its own style sheet, copy the parameters you need from the Blogs component's style sheet and insert them into your own site's style sheet. Make sure you resolve any conflicts between the Blogs component's style sheet and your own style sheet:

```
<render:callelement elementname="FW_Blogs/CSS/blogsCSS/">
```

The following lines are the JavaScript method that retrieves the **More** link for the Archive blogs page:

```
<script type="text/javascript">
function getMoreBlogs(url) {
  var xhtReq = getXMLHttpRequest();
  xhtReq.open(GET, url, true);
  xhtReq.onreadystatechange = function() {
    if(xhtReq.readyState==3) {
      document.getElementById('moreLink').innerHTML =
        ' <img src="<%=ics.GetSSVar("baseurl")%>images/wait_ax_tiny.gif/>';
    }
    else if(xhtReq.readyState==4) {
      document.getElementById('archiveDiv').innerHTML = xhtReq.responseText;
    }
  };
  xhtReq.send(null);
}

function getXMLHttpRequest() {
  try { return new XMLHttpRequest(); } catch(e) {}
  try { return new ActiveXObject("Msxml2.XMLHTTP"); } catch (e) {}
  alert("XMLHttpRequest not supported");
  return null;
}
</script>
```

The following lines call the main blog page or the body of a given asset. These lines also call the header and footer for the blog sample pages. Since your site has its own headers and footers, copy only the code you require for your site's blog functionality, and retain your own site's headers and footers:

```
<!--main start-->
<div id="main">
```

```
<!--header start-->
<%-- <div id="header">
<!--Call header from here-->
</div> --%>

<!--header end container start-->
<div id="container">
<!--Body -->
<%-- Call the container template for the current page subtype --%>
  <render:calltemplate
   tname='<%=sContainerTName%>'
   site='<%=sSite%>'
   tid='<%=ics.GetVar("tid")%>'
   slotname="BlogBodyContainer"
   c='<%=ics.GetVar("c")%>'
   cid='<%=ics.GetVar("cid")%>'
   ttype="Template">
      <render:argument name=p
       value='<%=ics.GetVar("p")%>' />
      <render:argument name=locale
       value='<%=ics.GetVar("locale")%>'/>
      <render:argument name=packedargs
       value='<%=ics.GetVar("packedargs")%>'/>
      <render:argument name=site
       value='<%=ics.GetVar("site")%>'/>
      <render:argument name=spubid
       value='<%=ics.GetVar("spubid")%>'/>
      <render:argument name=blogsperpage
       value='<%=ics.GetVar("blogsperpage")%>'/>
    </render:calltemplate>
  </div><!-- End of container -->
</div><!-- End of main -->

<!-- Footer -->
<%-- <div id="footer">
<!--Call footer from here-->
</div> --%>
</body>
```

6. Reconfigure the blog code you inserted into your site's layout template to fit your requirements.

7. Inspect the code of the other default blog templates and CSElements and copy the relevant sections into your own templates and CSElements.

52.2.3 Adding Blog Parameters to Your Site's SiteEntry Asset

The Blogs component's SiteEntry asset specifies the `blogsperpage` parameter, which enables you to specify the number of blog assets that can be displayed on a Web page at one time. You can specify this parameter in your own site's SiteEntry asset.

Note:

If you do not specify the `blogsperpage` parameter in your site's SiteEntry asset, then the default number of blogs per page, which is 10, will be used.

To add the blogsperpage parameter to your custom SiteEntry asset:

1. Log in to the Admin interface as a general administrator.

2. Select the site to which you are adding blog functionality.

3. In the button bar, click **Search** to find your site's SiteEntry asset.

   a. In the Search form, click **Find SiteEntry**.

   b. Click **Search**.

   c. Select your site's SiteEntry asset.

4. In the SiteEntry asset's Inspect form, click **Edit**.

5. In the **Pagelet Parameters** field, add the following:

   • **Name**: Enter blogsperpage.

   • **Value**: Enter the number of blogs that can be displayed on a page at one time.

6. Click **Save**.

52.3 Customizing URLs for the RSS Feed

The right navigation panel of the sample blog pages contains an RSS Feed link. When a visitor clicks the RSS Feed link, it renders an up-to-date listing of the titles and summaries of the blog assets that are published to the sample blog pages. When a visitor clicks the title of a blog, the entire content of the selected blog is rendered. External URLs for the blog assets included in the RSS Feed are created by the GetExternalURL CSElement.

By default this element creates the local WebCenter Sites URL:

```
http://<host name>:<port number>/<application context>/<path to file>
```

where <host name> is the host name of the WebCenter Sites installation from which the assets of the RSS Feed are accessible, <port number> is the port number of the WebCenter Sites application, and <application context> is the context of the WebCenter Sites application on which the Blogs component is running.

• To use the RSS Feed with your own site's external URLs, modify the GetExternalURL element's URL string, and add the parameters listed in the following table to the futuretense_xcel.ini file with the values for the host information and context of your site. The GetExternalURL element reads these parameters from the futuretense_xcel.ini file to create the external URLs for the blog assets listed in your site's RSS Feed.

Table 52-1   Parameters read by GetExternalURL element to create external URLs

Parameter	Description
fwblogs.hostscheme	Specifies the top level of the URL naming structure. For example, http.
fwblogs.hostname	Specifies the host name of the WebCenter Sites installation from which the assets of the RSS Feed are accessible.

Table 52-1   (Cont.) Parameters read by GetExternalURL element to create external URLs

Parameter	Description
`fwblogs.portnumber`	Specifies the port number of the WebCenter Sites application.
`fwblogs.contextinfo`	Specifies the context of the WebCenter Sites installation on which the Blogs component is running.

Part XIII

Developing WebCenter Sites: Visitor Services

You don't have to build Oracle WebCenter Sites: Visitor Services as an add-on. Read more to find out how you can configure Visitor Services.

- Developing WebCenter Sites: Visitor Services

53

Developing WebCenter Sites: Visitor Services

Oracle WebCenter Sites: Visitor Services fetches and compiles visitor profile information. Marketers use this information to orient their website content, such as product information and marketing campaigns, toward visitor needs.

Topics:

- Visitor Services Overview

- Configuring the Visitor Services URL

- Configuring an Identity Provider

- Configuring an Access Provider

- Configuring One or More Profile Providers

- Creating One or More Aggregation Templates

- Optimizing Experiences Using Visitor Services Data

- Visitor Services Reference

Note:

For information about Visitor Services-specific terms, see the Visitor Services Glossary.

53.1 Visitor Services Overview

Visitor's profile information is typically captured across different systems within customers' online presence. For example, a visitor may have recently viewed an Eloqua email, downloaded a CRM white paper, or logged into Facebook or Google. Each online activity collects a different set of visitor attributes. Visitor Services collects these visitor attributes and serves this information through an aggregate template.

The Visitor Services component provides discovery, aggregation, and query features that you can use for targeting. Choosing selected attributes from different visitor profiles, you can create as many aggregated templates as the marketers need. Marketers use the visitor information to target content on WebCenter Sites delivered pages.

Visitor Services Tasks for Developers

Visitor Services configuration primarily involves developer tasks organized on the Visitors Admin node of the Admin interface, as shown below. For more information

about Visitor Services from a marketer perspective, see Understanding WebCenter Sites: Visitor Services in *Oracle Fusion Middleware Using Oracle WebCenter Sites*.
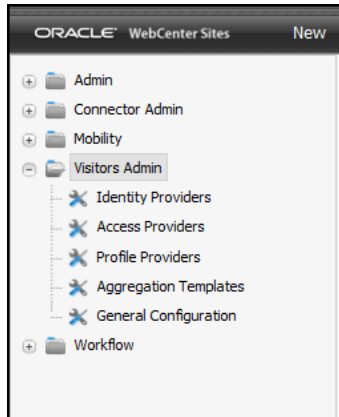
Developer tasks include:

- Configure the **Visitor Services URL** as described in Configuring the Visitor Services URL.

  To configure Visitor Services with WebCenter Sites, you need to identify the Visitor Services URL configured during installation. See Configuring the Visitor Services URL and Deploying Visitor Services in *Oracle Fusion Middleware Installing and Configuring Oracle WebCenter Sites*.

- Configure an **identity provider** for the single sign-on system (SSO) to authenticate site visitors to Visitor Services. See Configuring an Identity Provider.

  Visitor Services ships with an Oracle Access Manager identity provider for integration with Oracle Access Manager. You can also create a custom identity provider, as described in Creating a Custom Identity Provider: Example.

- Configure **access providers** to authenticate profile information requests. See Configuring Access Providers.

  An access provider qualifies the REST calls made from the application to Visitor Services. Oracle recommends using an access provider to maintain a secure connection between Visitor Services and WebCenter Sites.

- Create **profile providers** to compile and enrich visitor profiles.

  A profile provider allows a visitor identity to be associated with a visitor profile. You implement and configure profile providers for specified repositories in Visitor Services, and write enrichment rules to collect visitor information from all profile providers. Visitor Services ships with an Eloqua profile provider for integration with the Eloqua Cloud Marketing Service, an Oracle Access Manager profile provider for integration with Oracle Access Manager, a Facebook profile provider. See Configuring One or More Profile Providers and About the Profile Providers and Enrichment Service.

- Create **aggregate templates** that determine which data, based on visitor profiles, is returned to the site visitors. See Creating One or More Aggregation Templates and About the Identity Providers.

  An aggregation template links visitor profiles and lets you combine information from different visitor profiles. Developers write aggregation rules and templates that combine profile information from visitor profiles.

- Configure how visitor profiles from Visitor Services are **requested** and **used**.

  To make use of visitor profiles, WebCenter Sites components must request them via Visitor Services APIs (REST, JAVA, or JavaScript) during runtime. Visitor Services comes integrated with:

  - **Engage**: Use visitor profiles to determine Engage segments and provide recommendations. For example, a marketer might use visitor profile attributes such as age and income to create segments and deliver recommendations. For anonymous visitors, create segments for them based on attributes such as device used to access the site, time zone, locale, browser, referrer (Facebook, Twitter, Bing), or IP address. See Working with Engage Assets in *Oracle Fusion Middleware Using Oracle WebCenter Sites*.

  - **A/B Testing**: Use visitor profiles for determining segments, then use the segments to target visitors for the A/B test. See Working with A/B Testing in *Oracle Fusion Middleware Using Oracle WebCenter Sites*.

Visitor Services API References

- **Visitor Services Java API**: Provides Java API documentation for client developers of Visitor Services to write custom identity provider, access provider, and profile provider interface implementations.

- **Visitor Services Client Java API**: Provides Java API documentation for developers of applications such as Oracle WebCenter Sites and Engage that communicate with Visitor Services to get visitor profile information.

53.2 Configuring the Visitor Services URL

Before you can use Visitor Services, you need to configure its instance with your WebCenter Sites. The Property Management tool available in the Admin interface lets you quickly configure your Visitor Services instance.

1. Log into the Admin interface.

2. On the tree, expand the **Admin** node and the **System Tools** node.

3. Double-click the **Property Management** node.

   The Property Management Tool page is displayed.

4. From the **Category** drop-down list, choose **Visitors** and then click **Search**.

   The **Properties** section on the Property Management Tool page is populated with the properties for the Visitors category.

5. Under the **Key** column, click **visitors.rest.url.**

6. In the **Value** field, enter the Visitor Services instance URL in format: `http://<visitorserviceshost>:<visitorservicesport>/visitors-webapp.`

Figure 53-1 Properties section of the Property Management Tool Page

7. Click **Save** .

Your Visitor Services instance has been configured.

Figure 53-2 visitors.rest.url Property Configured in the Properties Section

53.3 Configuring an Identity Provider

An identity provider finds the status of visitor authentication to Visitor Services. You can configure an identity provider using the OSGi bundle or develop your own identity provider as an OSGi bundle.

- Using the Oracle Access Manager identity provider (OSGi bundle) provided with Visitor Services, then integrating Oracle Access Manager (OAM) with Visitor Services. See Configuring Identity Provider Settings and Integrating Oracle Access Manager (OAM) with Visitor Services.

- Developing your own custom identity provider as an OSGi bundle, then configuring it in Visitor Services. See Creating a Custom Identity Provider: Example.

Note that Visitor Services supports authenticated and unauthenticated visitors as follows:

- **Authenticated visitors**: These visitors are authenticated through identity providers. For example, a valid OAM user can be authenticated by an identity

provider created to identify those OAM users that reach Visitor Services via a
request call.

- **Unauthenticated/Anonymous visitors**: This type of visitor never signs in, so
  Visitor Services has no attributes for this user. While Visitor Services cannot
  identify anonymous visitors, if a client application (for example, WebCenter Sites)
  decides to register them explicitly, these visitors become authenticated visitors
  after their registration. The client applications can save attributes on their behalf
  post registration. The website can store the Visitor Services-generated visitor ID in
  a cookie and use it for associating the user sessions.

53.3.1 Configuring Identity Provider Settings

Whether you use the out of the box identity provider or create your own provider, the
identity provider's settings described in this topic must be set in the Admin interface.

To configure an identity provider:

1. Log into the Admin interface.

2. On the tree, expand the **Visitors Admin** node.

3. Double-click the **Identity Providers** option.

   The Identity Providers List page is displayed.

4. Click **Add New**.

   The Visitors Identity Providers form is displayed.

5. In the **Name** field, enter a meaningful name.

6. In the **Implementation** box, either drop the implementation OSGi bundle in the
   Drop File Here region, or click **Browse** to upload it. For example, to configure
   OAM identity provider, use the `OAMIdentityProvider` implementation
   provided with the Visitor Services application.

7. In the **Configuration** box, enter the configuration parameters of the identity
   provider you are creating. For example, if you are configuring OAM identity
   provider, the parameters are:

   - Visitor Identity Header name: `oam.headers.dn=OAM_REMOTE_USER`

   - Identity Storage Header name:
     `oam.headers.identityStorage=OAM_IDENTITY_DOMAIN`

   - Anonymous Visitor Identity Header value: `oam.guest=Anonymous`

Figure 53-3   Visitors Identity Providers Form
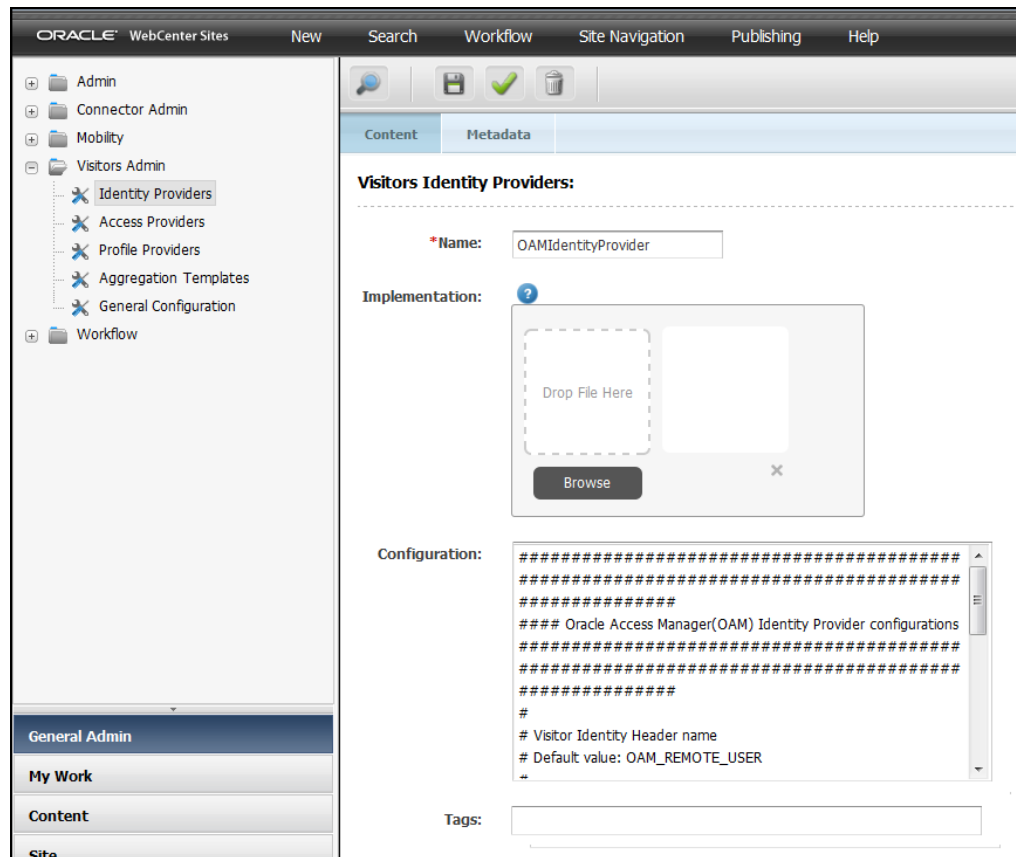
When the LDAP embedded with OAM is down and requests made through OAM do not reach Visitor Services, the Visitor Services application does not return any response about the LDAP's unavailability.

8. Click the **Save** icon.

The new identity provider is available on the Identity Providers List page.

To enable an identity provider for this site, first go to the Identity Providers List by double-clicking the **Identity Providers** node in the Admin tree on the left. Then, choose the **Enable** radio button for the provider you want to enable. At a time, only one identity provider can be enabled for a site.

53.3.2 Integrating Oracle Access Manager (OAM) with Visitor Services

Before performing steps described in this section, ensure that you have configured the OAMIdentityProvider provided with Visitor Services. The OAM identity provider enables Visitor Services to communicate with OAM. Steps to configure this provider are described in Implementing and Configuring Identity Providers.

After integrating OAM with Visitor Services as described in this section:

- Configure the OAM profile provider in Visitor Services. This profile provider will retrieve the actual data from the OAM storage. Steps to configure the profile provider are described in Implementing and Configuring Profile Providers and Enrichment Rules.

- Verify that the integration is successful.

Note:

The OAM-Visitor Services integration must be a standard implementation.

To integrate OAM Mobile & Social with Visitor Services:

1. Go to the OAM Admin console: `http://host:port/oamconsole`.

2. Click the **Application Domains** tab, then create an application domain.

Figure 53-4   Applications Domains
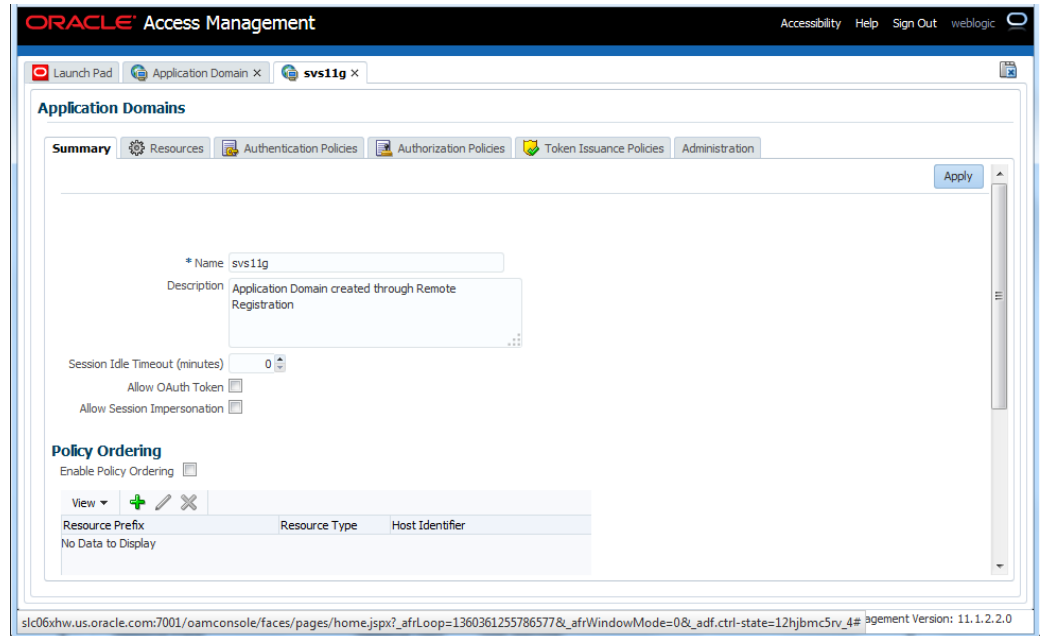
3. To create protection policies:

   - Under the Application Domains page, on the **Authentication Policies** tab, click **Create Authentication Policy**.

Figure 53-5   Authentication Policies
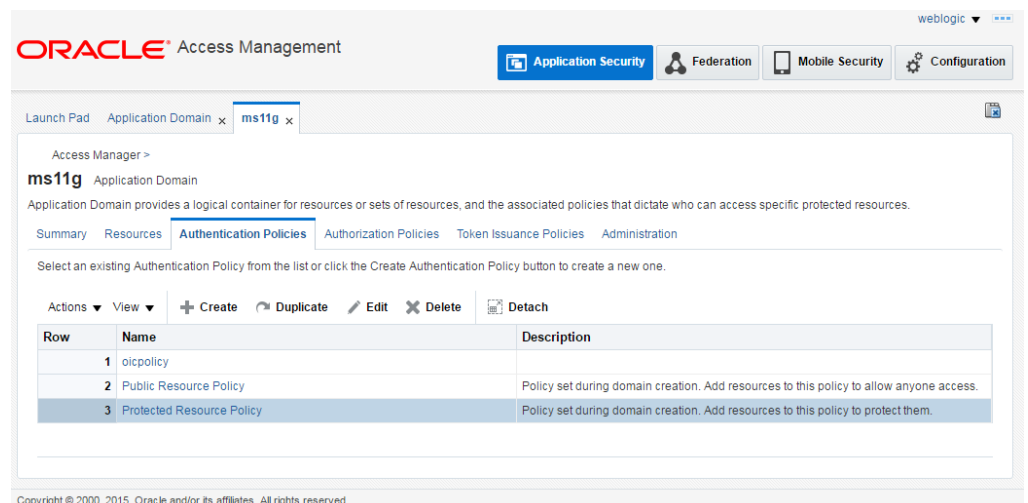
- On the Authentication Policy page, from the **Authentication Scheme** drop-down list, choose **OICSchema** and call it `oicpolicy`.

- Click **Add for Resources**, add `/index.html`, and then click **Apply**.

Figure 53-6   Authentication Policy - Authentication Scheme
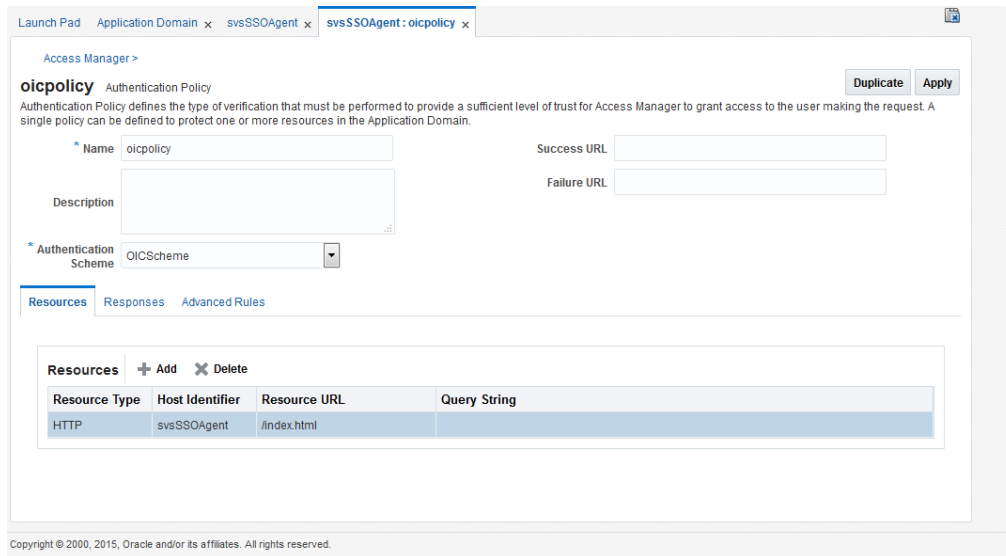
- On the Authentication Policy page, click **Protect Resource Policy**.

- From the Authentication Scheme drop-down list, choose **LDAPSchema** so that OAM uses LDAP as the AuthenticationModule.

- Click **Add for Resources**, add `/sites/avi/home.html` , `/**`. and then click **Apply**.
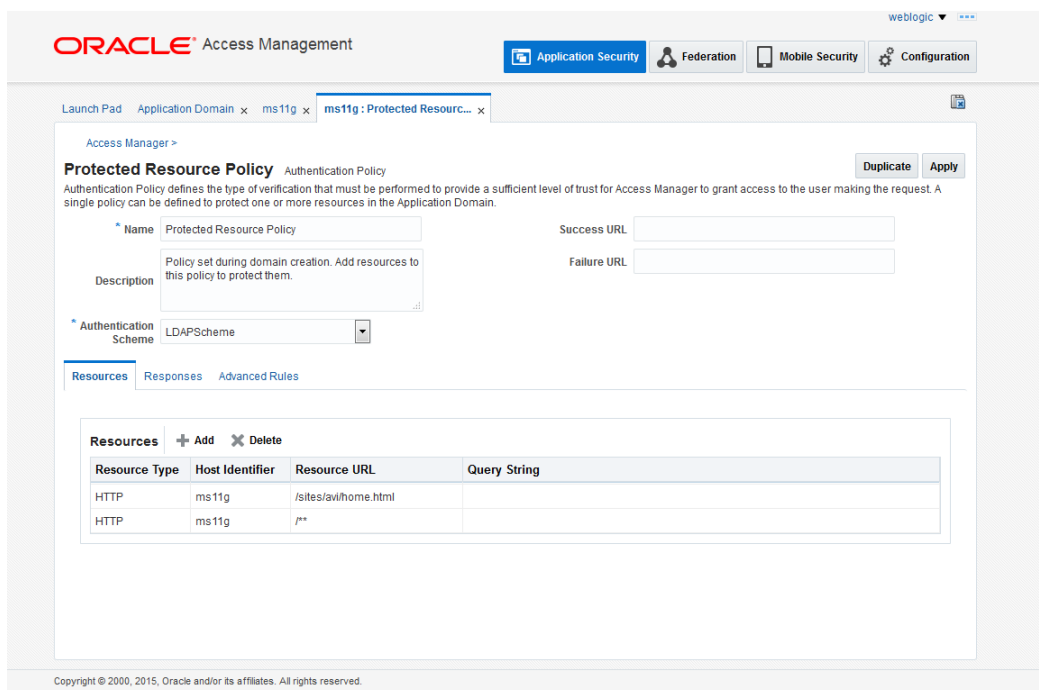
- From Launch Pad, open **Authentication Module**. Then, verify identity storage used for the LDAP module (see the administrator's guide for Oracle Access Manager). This storage is used when configuring the LDAP profile provider.

4. Under **Search**, from the **Resource Type** drop-down list, select **HTTP** and then click **Search**. Add the following resources if the search result doesn't return them.

- Add the following resources (see image below).

    - /index.html

    - /visitor-webapp/rest/*/visitor/**

    - /visitors-webapp/rest/*

    - /visitors-webapp/rest/*/visitor/current/**

    - /sites/avi/*.*

    - /visitors-webapp/**

    - /sites/avi/home.html

    - /**

- For **/visitors-webapp/rest/*/visitor/**, choose the protection level as **Excluded**.

- For **/visitors-webapp/rest/**, choose the protection level as **Unprotected**.

Figure 53-7   HTTP Resource Type
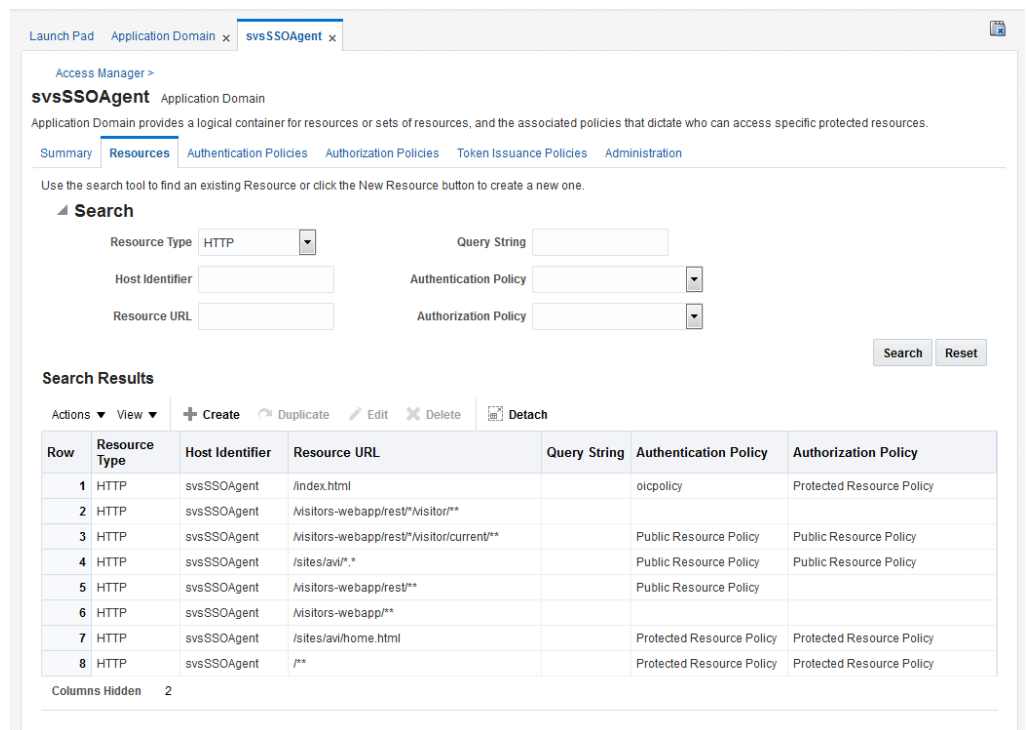
5. Create an SSO agent:

- On the Launch Pad, click **SSO Agent Registration**.

- Select **11g Webgate**.

- Complete the form.

Figure 53-8   svsSSOAgent
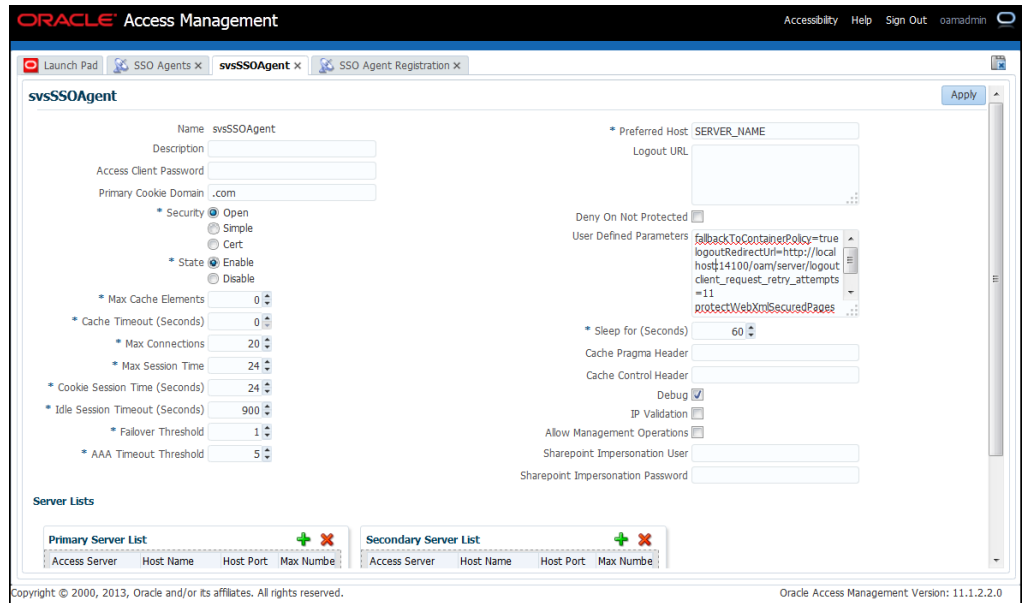
6. Configure Oracle Http Server (OHS):

   a. Go to Enterprise Manager: `http://host:port/em`.

   b. In the navigation tree, expand **Web Tier**, then select the OHS instance.

   c. To create a new virtual host, choose **Administration** and then choose **mod_wl_ohs Configuration** from the **Oracle Http Server** drop-down menu.

Figure 53-9   mod_wl_ohs Configuration
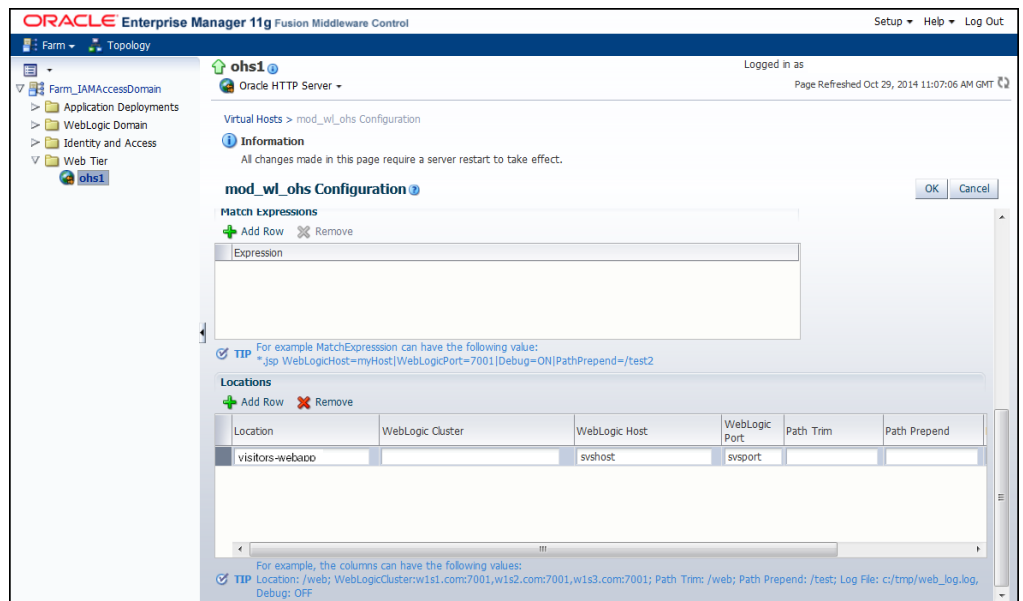
    d. Edit the `mod_wl_ohs.conf` file.

    e. In the section containing the **WebLogicCluster** directive, set
DynamicServerList to `OFF`.

    f. Save the `mod_wl_ohs.conf` file and restart OHS.

7. Configure Mobile and Social:

    a. From Launch Pad, open Social Identity.

    b. Create an application profile with the same name that you specified while
creating the application domain.

    c. Map application user attributes and Internet identity provider user attributes.

Figure 53-10   Application User Attribute and Internet Identity Provider User Attributes Mapping

Configure the profile provider, as described in Implementing and Configuring Profile
Providers and Enrichment Rules.

For your reference here is a sample code that lets you use the Java Script client with
OAM:

```html
<html>

<head>
    <title>Sample  Page</title>
    //Assuming OAM is at 'http://<host>:<port>',(Actual Visitor Service application
may be deployed somewhere else. Its location must be registered with OAM),
    //use OAM server in URLs for Visitor Services to go to the Visitor Services
application via OAM so that the identity token set by OAM is available to Visitor
Services.
    <script src="<host>:<port>/visitors-webapp/js/client.js"></script>
</head>

<body>

<script type="text/javascript">

    var handleSuccess = function( visitorId )
        {
         // Got visitorId
          console.log("visitor id received from visitor Services = "+visitorId);
        };

    var handleError = function(error)
     {
       conole.log(error.errorCode);
     };
```

```
     var client = new com.oracle.sites.visitors.Client("<host>:<port>/visitors-
webapp"); //URL having OAM instance as server part
     client.getVisitorId(handleSuccess,handleError);
</script>

</body>

</html>
```

Verify OAM-Visitor Services integration

1. Get a visitor Id using the Visitor Services Javascript client. For this, a page in WebCenter Sites, which is protected in OAM through policy 'protected', can be called via OHS so that the user is asked by OAM to log in. The use provides his credentials and lands on the page where the Javascript client code is present. The visitor ID received by the Javascript client is an OAM authenticated visitor ID. Use the OHS virtual host and port instead the host and port of Visitor Services. Do not specify any external ID because OAMIdentityProvider gets it from request. Generate visitor ID for each call.

2. Log into Visitor Services using an existing user.

3. Get the visitor ID as described in step 1. User ID should be generated on the first call, and it should be the same for each call.

4. Try to get the visitor's linked profiles using the Visitor Services REST API for the user generated in step 3.

53.3.3 Creating a Custom Identity Provider: Example

OAM identify provider is provided with Visitor Services out of the box. This section guides you to implement a custom identify provider with the help of an example.

The following points can guide you to implement a sample identity provider:

- Every identity provider must be an OSGi bundle, and this bundle should consist of the provider implementation and an activator to register the provider to Felix runtime.

- The implementation class should first implement the identity provider interface and then the profile provider interface. It must implement the identity provider interface with at least one of the following methods fully implemented:

    – Identity getVisitorIdentity(String externalId);

    – Identity getVisitorIdentity(HttpServletRequest request);

- The method shown in the following example returns the identity (just the Distinguished Name (dn)), which is passed to the profile provider to fetch the profile information.

```
package com.oracle.sites.visitors.api.providers;

import ...
public interface IdentityProvider
{
String getProviderName();
void setProviderConfig(Properties config);
Identity getVisitorIdentity(HTTPServletRequest request);
}
```

- The following is a sample code to implement an identity provider. This implementation uses an external Id consisting of a provider name and a user dn:

```
public class SampleIdentityProvider implements IdentityProvider
{
     private static final String PARAMETER_EXTERNAL_ID_NAME =
"parameter.external_id";
     private static final String COOKIE_NAME_ATTRIBUTE_NAME =
"cookie.external_id";

     @override
     public Identity getVisitorIdentity)HttpServletRequest request)
     {
         String externalId = getExternalIdFromCookie(request);

         if (null ==externalId || externalId.isEmpty())
         {
             externalId = getExternalIdFromParams(request);
         }
                 //Returns an identity object based on external id
         return getVisitorIdentity(externalId);
     }
```

The system calls the profile provider with the identity object returned above. Every time a user requests for the visitor Id, the external id (provider and dn) is stored in the cookie for the future use.

- OSGi bundles require an activator class to activate the code contained in the bundles. Write an activator class that can register the sample identity provider.

```
public class Activator implements BundleActivator {
    /**
     * Implements BundleActivator.start(). Prints
     * a message and adds itself to the bundle context as a service listener.
     * @param context the framework context for the bundle.
    **/
    public void start(BundleContext context)
    {
        context.registerService (ProfileProvider.class.getName(),
                new SampleProfileProvider(context.getBundle().getLocation()),
null);
    }

    /**
     *Implements BundleActivator.stop().Prints a message
     *and removes itself from the bundle context as a service listener.
     *@param context the framework context for the bundle.
    **/
    public void stop(BundleContext context)
    {
    }
}
```

- Configure the identity provider by following the steps described in Configuring an Identity Provider.

53.4 Configuring an Access Provider

An access provider controls applications' access to visitor profile information. The access provider service works in pair with the REST client. For example, for the container protection authentication you should upload the access provider bundle

with basic authentication settings on Visitor Services, and set the same authentication type on the REST client.

See Access Provider Reference.

Visitor Services ships with a basic LDAP access provider for authentication against an LDAP directory.

Follow the steps in this section to enable an access provider to use with Visitor Services:

To configure an access provider:

1. Log in to the Admin interface.

2. On the tree, expand the **Visitors Admin** node.

3. Double-click the **Access Providers** option.

   The Access Providers List page is displayed.

4. Click **Add New**.

   The Visitors Access Providers form is displayed.

   Figure 53-11   Visitors Access Providers Form

5. In the **Name** field, enter a meaningful name.

6. In the **Implementation** box, either drop the implementation OSGi bundle (For example, `ContainerBasicAccessProvider`) in the **Drop File Here** region, or click **Browse** to upload it.

The access provider is a OSGi bundle containing the files required by Visitor Services. This OSGi bundle is an actual JAVA-based implementation of the communication required between Visitor Services and the desired storage. The access provider OSGi bundle consists of the following:

- The `Activator` class which is the entry point of the OSGi framework.

- The `MANIFEST.mf` file with OSGi Headers described in this table:

Table 53-1   OSGi Headers in MANIFEST.mf File

Header Name	Description
Bundle-Name	Name of the access provider.
Bundle-SymbolicName	`com.sample`
Bundle-Description	Description of the access provider
Bundle-ManifestVersion	`1`
Bundle-Version	`1.0.0`
Bundle-Activator	`com.sample.Activator`

Import-Package	`org.osgi.framework;version=4.2.1,org.json,`
	`com.oracle.sites.visitors.api.providers,`
	`com.oracle.sites.visitors.api.providers.beans`

- The name of the configuration file must be the same: `accessProviderConfig.properties`. This file must contain `key=value` pairs of properties required by current implementation of the access provider. The configuration file must always exist in the OSGi bundle, even though blank.

7. In the **Configuration** box, enter the configuration of the access provider you are creating. For example, for `ContainerBasicAccessProvider`, enter `role=role Configured In Container for Visitors Services`.

   If `Sitewrapperapi` is used, then you must also set the following properties in the Property Management Tool: `wcsites.visitors.auth.password`, `visitors.rest.authalias`, `visitors.rest.authtype`, `visitors.rest.authheader`.

8. Click the **Save** icon.

   The new access provider is available on the Visitors Access Providers page.

9. To enable this access provider for this site, go to the Access Providers List by double-clicking the **Access Providers** node in the Admin tree on the left. Then, choose the **Enable** radio button for the provider you want to enable. At a time, only one access provider can be enabled for a site.

53.5 Configuring One or More Profile Providers

A profile provider lets you associate a visitor identity with a visitor profile. Visitor Services ships with an Eloqua profile provider that you integrate with Eloqua, an Oracle Access Manager profile provider for Oracle Access Manager, and a sample profile provider.

See Profile Provider Reference.

You can configure a profile provider by:

- Using the Eloqua provided with Visitor Services. See About Configuring Eloqua Profile Provider.

- Using the Oracle Access Manager profile provider (OSGi bundle) provided with Visitor Services. See Configuring Profile Provider Settings and Enrichment Rules.

- Developing your own custom profile provider as an OSGi bundle, then configuring it in Visitor Services. See Creating a Custom Profile Provider: Example.

Note: Do not change the attribute names of the profile providers once these attributes are set.

53.5.1 Configuring Profile Provider Settings and Enrichment Rules

To configure a profile provider and enrichment rules:

1. Log in to the Admin interface.

2. On the tree, expand the **Visitors Admin** node.

3. Double-click the **Profile Providers** option.

   The Profile Providers List page is displayed.

   Figure 53-12   Profile Providers List

4. Click **Add New**.

   The Visitors Profile Providers form is displayed.

5. In the **Name** field, enter a meaningful name for the profile provider you are configuring. In case of the OAM identity provider, the name of the profile provider should be the same as that of the embedded LDAP store.

6. Next to the **Implementation** field, click **Browse** to upload the profile provider OSGi bundle. If LDAP is used, then upload the LDAP profile provider provided with Visitor Services, otherwise use your custom implementation.

   The profile provider OSGi bundle consists of the following:

   • The `Activator` class which is the entry point of the OSGi framework.

   • The `MANIFEST.mf` file with OSGi Headers.

   Table 53-2   OSGi Headers in MANIFEST.mf File

Header Name	Description
Bundle-Name	Name of the profile provider.
Bundle-SymbolicName	`com.sample`
Bundle-Description	Description of the profile provider
Bundle-ManifestVersion	`1`
Bundle-Version	`1.0.0`
Bundle-Activator	`com.sample.Activator`

Import-Package	`org.osgi.framework;version=4.2`
	`.1,org.json,`
	`com.oracle.sites.visitors.api.`
	`providers,`
	`com.oracle.sites.visitors.api.`
	`providers.beans`

   • The name of the configuration file must be the same: `profileProviderConfig.properties`. This file must contain `key=value` pairs of properties required by current implementation of the profile provider. The configuration file must always exist in the OSGi bundle even if the file is blank.

7. In the **Configuration** box, enter configuration parameters for your profile provider. If you are using the LDAP profile provider, see this table:

   Table 53-3   LDAP Profile Provider Configuration Parameters

Parameter	Value
LDAP Address	Format: `ldap://<host>:<port>/`.
Base Distinguished Name for users	`BaseUserDN=cn=Users,dc=com`
Distinguished Name of the manager (admin)	`AdminDN=cn=admin,dc=com`

Table 53-3 (Cont.) LDAP Profile Provider Configuration Parameters

Parameter	Value
Password	Password of the manager (admin). `AdminPassword=password`
LDAP attribute used as the user login name	Default is `uid`. `LoginUserAttribute=uid`.
Supported LDAP objectClasses	`ObjectClasses=top;inetOrgPerson`
LDAP profile attributes	`ProfileAttributes=cn,sn,displayName,mail,t itle,givenName,name`

Figure 53-13 Visitors Profile Providers Form
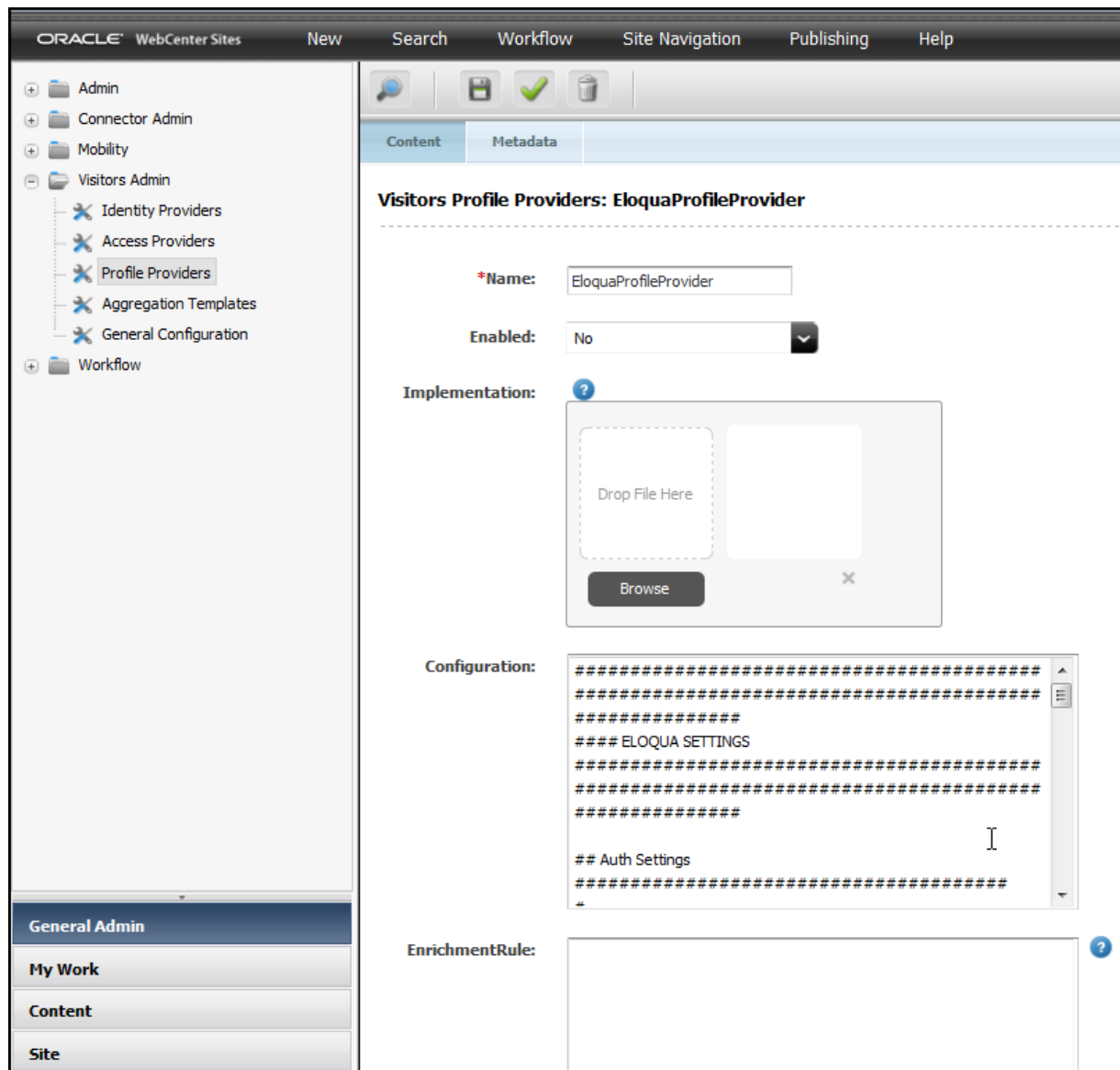
8. In the **Enrichment Rule** box, enter enrichment rules you wish to apply on visitor profiles. Enrichment rules includes attributes that define visitors profile. These

rules are based on multiple visitor profiles available in different storages, as well as arbitrary attributes that customers wish to add to visitor data that Visitor Services will be collecting and providing to applications.

To configure enrichment rules for LDAP, CRM, and Eloqua profile providers, `dn` used in LDAP profile provider must be the same as the `name` in the CRM profile provider, the CRM profile provider includes the `email` attribute, and the Eloqua profile provider includes the `Id` attribute. Thus, to allow enrichment process to run, profile providers should be configured as follows:

- Add the following enrichment rule to the LDAP profile provider:
  `dn=CRM:name`

- Add the following enrichment rules in CRM profile provider:

  - `name=LDAP:dn`

  - `Id=ELOQUA:Id`

- You can write the enrichment rule for the Eloqua profile provider based on the `Id` attribute. For example:

  Eloqua enrichment rule for LDAP:

  `Id=eloqua:Id`

  Eloqua enrichment rule for unknown profile providers:

  `dn=eloqua:id`

Consider the following:

- In case of LDAP profile provider, the LDAP embedded with OAM does not support search for some attributes such as homePhone, employeetype, departmentnumber, homepostaladdress, c, l, employeenumber. Therefore, enrichment rules based on these attributes may not work.

- Oracle Virtual Directory's search functionality supports only those attributes that use ASCII characters.

9. Click the **Save** icon.

   A new row is created in the Providers table. The Visitor Services log is updated confirming that the new provider has been added.

   The new profile provider is available for editing on the Profile Providers List page.

10. To enable this profile provider, go to the Profile Providers List by double-clicking the **Profile Providers** node in the Admin tree on the left. Then, select **Enable** check boxes for the providers you want to enable. You can enable multiple profile providers simultaneously.

53.5.1.1 About Configuring Eloqua Profile Provider

To use the Eloqua profile provider, you must:

1. Follow the procedure described in Configuring Profile Provider Settings and Enrichment Rules to create a profile provider for Eloqua.

   Ensure that you enter the details of the Eloqua provider instance in the **Configuration** box.

For example:

```
#############################################################################
#################
#### ELOQUA SETTINGS
#############################################################################
##################

## Auth Settings
########################################
#
# Client's Company name
#
company = OraclePOC

#
# Client's user name
#
user =  username

#
# Client's user password
#
password =  password


## Proxy Settings
########################################
#
# Use proxy in provider
#
useProxy = true

#
# Proxy server host
#
proxyHost = www-proxy.us.oracle.com

#
# Proxy server port
#
proxyPort = 80

#
# Proxy server type
# This can be "DIRECT", "HTTP", or "SOCKS"
#
proxyType = HTTP

## Search Settings
########################################
#
# Eloqua REST URL
#
restUrl = https://secure.eloqua.com/API/REST/1.0

#
# Depth or level of visitor detail returned.
# This can be "minimal", "partial", or "complete".
#
searchDepth = partial
```

2. To enable the Eloqua Profile Provider profile provider, go to the Profile Providers List by double-clicking the **Profile Providers** node in the Admin tree on the left. Then, select **Enable** check boxes for **Eloqua Profile Provider** .

> **Note:** Eloqua does not support email-based search. Therefore, enrichment rules based on the email field in Eloqua do not work. Only Id-based search is supported. Ensure that the identity provider is configured in such a way that it identifies the user Id in Eloqua so the profile provider can fetch visitor profile details.

53.5.2 Creating a Custom Profile Provider: Example

Visitor Services comes packaged with LDAP profile provider, Eloqua profile provider, and a sample profile provider. The following points can guide you to implement a custom CSV profile provider:

- Every provider must be an OSGi bundle, and this bundle should consist of the provider implementation and an activator to register the provider to Felix runtime.

- The implementation class should first implement the identity provider interface and then the profile provider interface.

- The profile provider method shown in the following example fetches the profile information.

```
public interface ProfileProvider
{
String getProviderName();
RawProfile getProfile(String dn);
List<RawProfile> search(String attribute, String value);
void setProviderConfig(Properties config);
```

- Write an activator class that can register the sample profile provider:

```
public class Activator implements BundleActivator {
    /**
     * Implements BundleActivator.start(). Prints
     * a message and adds itself to the bundle context as a service listener.
     * @param context the framework context for the bundle.
    **/
    public void start(BundleContext context)
    {
        context.registerService (ProfileProvider.class.getName(),
                new SampleProfileProvider(context.getBundle().getLocation()),
null);
    }

    /**
     *Implements BundleActivator.stop().Prints a message
     *and removes itself from the bundle context as a service listener.
     *@param context the framework context for the bundle.
    **/
    public void stop(BundleContext context)
    {
    }
}
```

- In the following sample implementation, the `dn` passed for the identity object can be used to get the raw profile information. The profiles are loaded from `*.csv` file.

```
ID,Name,Email,Address,Country,City
1,Alek,Alek.Z@mycompany.com,Kyiv,Ukraine,Kyiv
2,Avi,avi.n@mycompany.com,Gachibowli,India,Hyderabad
3,Joe,Joe.d@mycompany.com,Miyapur,India,Hyderabad
4,Reva,reva.p@mycompany.com,Kukatpally,India,Hyderabad


@Override
public RawProfile getProfile(String dn) {
    List<CSVProfile> profiles = csvManager.getProfiles();
    CSVProfile selected = null;
    for (CSVProfile p : profiles) {
        if (p.getName().equalsIgnoreCase(dn)) {
            selected = p;
            break;
        }
    }

    if (selected == null)
        return null;
    RawProfile rawProfile = new RawProfile();
    rawProfile.setUserId(selected.getId());
    rawProfile.setProviderId(getProviderName());
    rawProfile.setAttributes (getAttributes (selected));
    return rawProfile;
    }

    private JsonObject getAttributes(CSVProfile csvProfile)
    {
        JsonObject attributes = new JsonObject();

        attributes.addProperty("email", csvProfile.getEmail());
        attributes.addProperty("address", csvProfile.getAddress());
        attributes.addProperty("country", csvProfile.getCountry());
        attributes.addProperty("city", csvProfile.getCity());
        return attributes;
    }
```

- Write an activator class that can register the sample profile provider:

```
public class Activator implements BundleActivator {
    @Override
    public void start(BundleContext bundleContext) throws Exception {
         bundleContext.registerService(ProfileProvider.class.getName(),
                   new
CSVProfileProvider(bundleContext.getBundle().getLocation()), null);
    }

    @Override
    public void stop(BundleContext bundleContext) throws Exception {

    }
}
```

- Configure the profile provider by following the steps described in .

53.6 Creating One or More Aggregation Templates

Based on visitor profiles, an aggregation template helps determine what information should be sent to the site visitors. For example, you can call an aggregation template from Engage to identify the attributes that you want to use for recommendations. You can create aggregation templates using Velocity or JavaScript.

See Aggregation Template Reference.

To configure aggregation templates:

1. Log in to the Admin interface.

2. On the tree, expand the **Visitors Admin** node.

3. Double-click the **Aggregation Templates** option.

   The Aggregation Templates List page is displayed.

   Figure 53-14   Aggregation Templates List
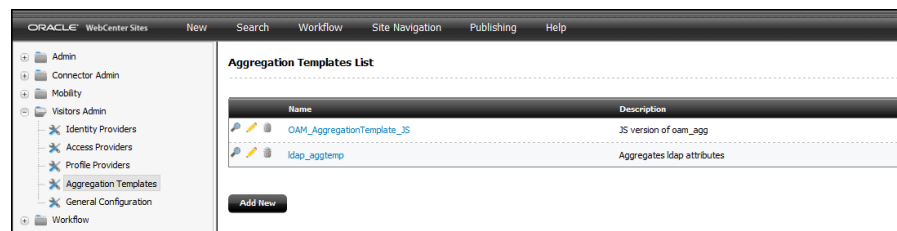
4. Click **Add New**.

   The Visitors Aggregation Templates form is displayed.

5. In the **Name** field, enter a meaningful name for the template you are configuring.

6. In the **Language** field, choose either **Velocity** or **JavaScript**.

7. In the **Template** box, enter aggregation template code. The following example shows a sample aggregation template that compiles visitor information from multiple profile storages. This sample is a Velocity code.

   Sample Velocity Code for Creating an Aggregation Template

```
#set( $sn ="Not Provided")
     #if(${profiles.LDAPProfileProvider.get(0).sn} != "null")
     #set( $sn = ${profiles.LDAPProfileProvider.get(0).sn.getAsString()})
     #end

     #set( $displayName ="Not Provided")
     #if(${profiles.LDAPProfileProvider.get(0).displayName} != "null")
          #set( $displayName = $
{profiles.LDAPProfileProvider.get(0).displayName.getAsString()})
       #end

     #set( $mail ="Not Provided")
     #if(${profiles.LDAPProfileProvider.get(0).mail} != "null")
          #set( $mail = $
{profiles.LDAPProfileProvider.get(0).mail.getAsString()})
       #end
```

```
          #set( $homePhone ="Not Provided")
          #if(${profiles.LDAPProfileProvider.get(0).homePhone} != "null")
                  #set( $homePhone = $
{profiles.LDAPProfileProvider.get(0).homePhone.getAsString()})
          #end

          #set( $name ="Not Provided")
          #if(${profiles.LDAPProfileProvider.get(0).name} != "null")
                  #set( $name = $
{profiles.LDAPProfileProvider.get(0).name.getAsString()})
          #end

          #set( $description ="Not Provided")
          #if(${profiles.LDAPProfileProvider.get(0).description} != "null")
                  #set( $description = $
{profiles.LDAPProfileProvider.get(0).description.getAsString()})
          #end

#else
        #set( $otherProvider = "other profile provider used.")
#end

#if (${profiles.has("LDAPProfileProvider")})
{
"Name ": "$name",
"displayName ": "$displayName",
"description ": "$description",
"sn ": "$sn",
"mail ": "$mail",
"homePhone ": "$homePhone"
}
#else
{
"otherProvider ": "$otherProvider"
}
#end
```

The following example shows a sample aggregation template that compiles visitor information from multiple profile storages. This sample is a Javascript code that does the same as the Velocity code above:

Sample JavaScript Code for Creating an Aggregation Template

```
function aggregate()
{
    var result={};
    if(profiles.get("LDAPProfileProvider"))
    {
        var NOT_PROVIDED_LABEL = "Not Provided";
        var internalAttrs =
JSON.parse(profiles.get("LDAPProfileProvider").get(0).getAttributes());

        result.Name = internalAttrs.name || NOT_PROVIDED_LABEL;
        result.displayName = internalAttrs.displayName || NOT_PROVIDED_LABEL;
        result.description = internalAttrs.description || NOT_PROVIDED_LABEL;
        result.sn = internalAttrs.sn || NOT_PROVIDED_LABEL;
        result.mail =internalAttrs.mail || NOT_PROVIDED_LABEL;
        result.homePhone = internalAttrs.homePhone || NOT_PROVIDED_LABEL;
```

```
        }
        else
        {
            result.otherProvider = "other profile provider used.";
        }
        return result;
    }
```

Figure 53-15   Visitors Aggregation Templates Form

8. Click the **Save** icon.

The new aggregation template is available on the Aggregation Templates page.

53.7 Optimizing Experiences Using Visitor Services Data

To help you optimize visitors' experience, extended attributes and activities collect visitor information from multiple profiles of a visitor. You can also integrate Visitor Services with Engage. In WebCenter Sites, page authors use this information to design segments and recommendations to display relevant content to the targeted users. Marketers use Visitor Services data for targeting, testing, and analysis.

Requesting visitor information involves two steps:

*   Requesting the visitor ID.

*   Requesting the visitor profile using a specified aggregation template.

    Returns the aggregated template and specified attributes.
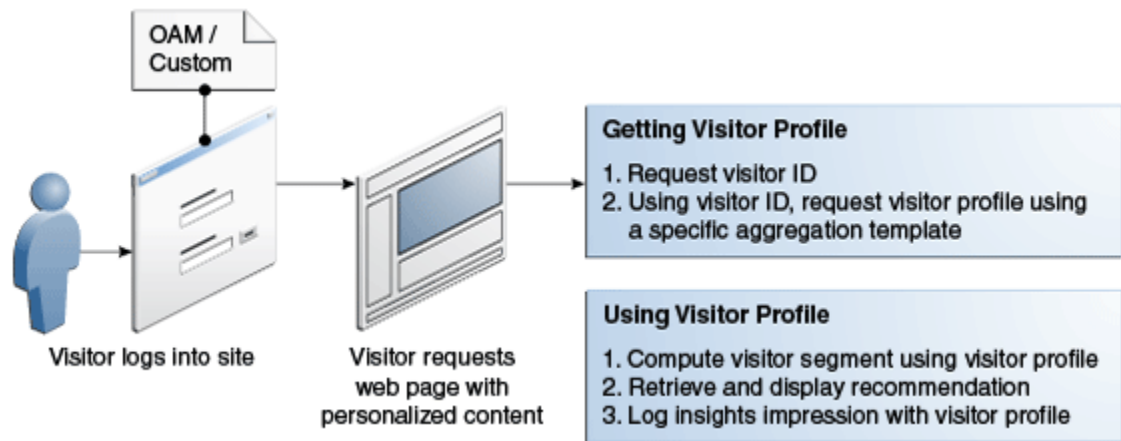
This section includes the following topics:

- How WebCenter Sites Components Request Visitor Services Profile Information

- Configuring Visitor Services with Engage

- **Storing Additional Information with Extended Attributes and Activities**

53.7.1 How WebCenter Sites Components Request Visitor Services Profile Information

To enable communication between Visitor Services and client applications such as WebCenter Sites, Visitor Services provides client APIs (Java and JavaScript) that are executed in client applications to hit Visitor Services. Out of the box, Visitor Services provides Java client for the server side and JavaScript client for the client side. Java client is used when Visitor Services and applications try to access profile information from server side. JavaScript client APIs are used when visitors, using applications from browsers, try to communicate with Visitor Services. The difference between the Java Client API and the JavaScript Client API is that the Java Client API is expected to be running inside trusted applications such as the server side code in WebCenter Sites. Therefore, these applications have the credentials available to authenticate themselves before processing some sensitive operations on the visitor data (update operation). The JavaScript Client executes inside a browser, and it is less secured. Therefore, it can read the current visitor's information only. Visitor Services provides information about all the visitors' profiles to trusted applications. To visitors, it provides information only about themselves.

Java and JavaScript client APIs have these functions in common. However, in case of JavaScript the functions are for the current/logged in visitors.

- `GetVisitorId`: Gets the visitor's ID from Visitor Services using the external ID stored in the SSO solution. In JavaScript APIs, this function is called `GetCurrentVisitorId`. It gets the visitor ID for the current user from Visitor Services.

- `GetAggregatedProfile`: Gets the aggregated profile from Visitor Services using the existing visitor's ID. In JavaScript APIs, this function is called `GetCurrentAggregatedProfile`. It gets the aggregated profile for the current user from Visitor Services.

  You can set profile expiration period from 0 day to any positive number (1 = 1 day). The expiration period determines how many days the profile information fetched from external storage should remain in the Visitor Services database, until this information is refreshed. If you do not specify an expiration period, then the

profile information is not refreshed at all. The General Configuration page, which is accessed through the **General Configuration** node under the **Visitors Admin** node, contains the **Profile Update Period** configuration. Edit this configuration to set the expiration period.

- `IsExists`: Verifies if the visitor ID exists in the database. Whether a guest or a registered user with some profile exists or not, the response will be `true` if the visitor Id exists in the database. In JavaScript APIs, this function is called `IsCurrentExists`. It verifies if the visitor ID exists in the database. Whether a guest or a registered user with some profile exists in the database or not, the response will be `true` if the visitor Id exists in the database.

- `IsGuest`: Verifies that visitor has no profiles in the database. Whether the visitor information exists or not, the response will be `true` in both cases. In JavaScript APIs, this function is called `IsCurrentGuest`. It verifies that the no visitor profile for this guest visitor exists in the database. Whether the guest visitor is stored in the database or not, the response will be `true` in both cases.

Java client is run at the server, while JavaScript is executed from the web browser. The URLs for both are differently structured. For example:

- Java REST function for trusted applications: `http://
visitorserviceshost:visitorservicesport/visitors-webapp/
rest/v1/visitor/id/690b507b-efb6-4c35-9e6e-0c9f2a28b37d/
profile/aggregated/arrg`

    In this URL notice `id/690b507b-efb6-4c35-9e6e-0c9f2a28b37d`. This is the specified visitor ID (for the logged-in visitor, `current` is used in its place, as shown in the next point.)

- JavaScript REST function for non-trusted visitors: `http://
visitorserviceshost:visitorservicesport/visitors-webapp/
rest/v1/visitor/current/profile/aggregated/arrg`

    In the above URL the visitor Id path is replaced with the `current` context which indicates that untrusted visitor can information only about himself. All REST calls for untrusted customer include the `current` context.

When all applications are deployed on OAM, Oracle HTTP Protection is set to the **excluded** security type. If a client application requests for profile information for which the data is already prepared, Visitor Services Java client APIs' Send request (through which already prepared data is provided to these applications) can be protected through Visitor Services' own protection:

- Container Protection: All the information about users is stored in the container storage, and Visitor Services only checks for users' roles. Oracle recommends using the container protection if client's web server allows basic authentication.

- Visitor Services Protection: Use Visitor Services protection when a client's web server doesn't support the basic authentication mechanism.

You can also implement your own access provider using Visitor Services API and upload your provider through the Admin interface.

> **Note:** When a request for an aggregated or linked profile is processed with the `updated` param set to `false`, the behavior of Visitor Services for expired profiles is as follows:
>
> - JMS configured for Visitor Services: The current request returns old profile data while Visitor Services updates the profile in the background to refresh the profile data in sometime.
>
> - JMS not configured for Visitor Services: Returns the updated profile in the request call since the background processing capability is dependent on JMS.

It is recommended that each application has its own credentials for accessing REST APIs. Visitor Services verifies this access by itself. Visitor Services REST APIs for untrusted applications are secured by SSO, and in OAM the protection level is set to **unprotected**. Visitor Services has no protected REST resources. Oracle recommends using the Container protection.

See the *Oracle Fusion Middleware Java API Reference for Oracle WebCenter Sites: Visitor Services*.

53.7.2 Configuring Visitor Services with Engage

You can integrate Visitor Services with Engage to collect and provide visitor information to Engage for use in segments and recommendations. In WebCenter Sites, page authors use this information to design segments and recommendations to display relevant content to the targeted users. To get profile information from Visitor Services, developers or page authors use the Value Source field in the Visitor Attributes form which is designed to collect and store visitor information as attributes.

1. In the Properties Management tool, ensure that Visitor Services properties are set, as described in Oracle WebCenter Sites: Visitor Services Properties in *Oracle Fusion Middleware Property Files Reference for Oracle WebCenter Sites*.

2. On the Visitor Services node in the Admin interface, configure the aggregation template to be used in the recommendation or segmenting. See Creating One or More Aggregation Templates.

3. In the Contributor interface, configure the **Value Source** field in the Visitor Attributes form, using Expression Language (EL) to reference the data in the aggregated template to populate the value of the visitor attribute. The visitor attribute is then used for segmenting and evaluating recommendations. For example, an EL expression in the **Value Source** field might look like this, where `Agg_Temp` is an aggregated template that collects the visitors' addresses, street numbers/name, and house address, `AnotherTemplate` identifies a second aggregated template, `My City` is the city in which a visitor lives, and `Zip Code` is the city's zip code: `${Agg_Temp.Address.PostalCode},${AnotherAggregatedTemplate["My City"].ZipCode}`. For more information about EL, see https://docs.oracle.com/javaee/6/tutorial/doc/gjddd.html.

4. In the Recommendation Reader API, create an implementation object that includes the same attributes set up in the aggregation template. During execution, the Recommendation Reader API interacts with Visitor Services to obtain the aggregated template using the following methods:

- **readSegments**: Determines the Engage segments to which the user belongs, after evaluating the visitor attributes provided or computed for the user. This method returns a list of map items where the key/value pairs in the map items describe the segments.

- **retainSegments**: Determines the user's membership in the specified Segments, after evaluating the visitor attributes provided or computed for the user. This method returns a list of map items where the key/value pairs in the map items describe the segments. This method is similar to the readSegments method, but it only considers the supplied list of segments, not all available segments.

- **readRecommendations**: Evaluates the named Engage Recommendation based on the visitor attributes provided or computed for the user. This method returns a list of map items where the key/value pairs in the map describe the assets produced by the recommendation.

5. The readSegments, retainSegments, and readRecommendations APIs set the `svsvisitorid` cookie if metadata used to address the call comes from Visitor Services. No cookie is set if the segments or recommendations can be computed without the use of Visitor Services. If the user has not logged in to Visitor Services, then `svsvisitorid` is set to a guest Id. This guest Id is used for subsequent segment or recommendation computation calls. If the user logs in, the developer needs to set the correct `svsvisitorid` cookie for that user. Similarly, when the user logs out, the`svsvisitorid` cookie may need to be reset.

53.7.3 Linking Visitor Profiles and Managing Cookies

You can use the `oracle.fatwire.integrations.svs.VisitorServiceHelper` class to get the current visitor's profile information and to manage visitor cookie lifecycles.

Suppose your website contains a Home page and a My Account page. If a visitor has not logged into the Home page using the Visitor Services identity provider service, the Helper API code (`helper.getProfile(ics,aggregationTemplateName,..)`) inside the page sets a guest ID in the browser as a cookie. (Note that a guest ID has no profile information.) If a link to the My Account page is protected through an authentication mechanism and the visitor logs in using the identity provider service before coming to the My Account page, then the visitor becomes an authenticated visitor. However, because the Helper API already set the guest ID in the browser cookie in the Home page, the cookie in the browser needs to be updated, by passing the `updateNow` param as `true` (`helper.getProfile(ics,aggregationTemplateName,true)`) inside the My Account page code. If the cookie is not updated, the My Account page won't receive the authenticated visitor ID and the related profile information, and it will keep using the guest visitor ID from the cookie.

In this example, the My Account page uses the following code:

```
VisitorServiceHelper helper = VisitorServiceHelper.getInstance();
Helper. getProfile(ics,<someAggregationTemplateName>,true);
```

In the above code, the behavior of the true value for the Boolean param is as follows:

- If the visitor ID from the browser cookie is null, it indicates that the visitor landed directly on an authenticated page. Otherwise, it indicates that it is a guest ID: that the ID does not exist in Visitor Services and was cached in the local browser only. In this case, generate the new visitor ID and replace the value in the cookie.

- If the visitor ID from the cookie exists in Visitor Services, keep using this ID, as the cookie links the two IDs. This is what happens when the guest ID in the cookie is registered explicitly before it is used in the My Account page. The code in the Home page either calls this API passing the Boolean param as false or calls another method that takes only two parameters: Helper. `getProfile(ics,<someAggregationTemplateName>)`, where this parameter is assumed to be false by default.

53.7.4 Storing Additional Information with Extended Attributes and Activities

This section describes extended attributes and activities, and it explains how they work in Visitor Services. This section includes the following:

- About Extended Attributes and Activities

- **How to Use Extended Attributes and Activities in Visitor Services**

53.7.4.1 About Extended Attributes and Activities

Extended attributes are based on visitors' characteristics such as, language, locale, gender, and so on. Extended attributes relate to a visitor by the visitor Id and include name and value. These attributes are specific to the applications. With an application's approval, these attributes are stored in Visitor Services and used by other applications. For instance, extended attributes from Engage, if approved, can be used by any other application.

Extended activities are based on a visitor's actions on a website. For example, opening a page, performing searches, bids etc. Activities relate to a visitor by the visitor Id and include type, data, timestamp, and rating. These activities are specific to the applications. With an application's approval, these activities are stored in Visitor Services and used by other applications.

Extended activities and attributes are collected from all the profiles of the visitors. That is, if a visitor has more than one linked profile, then the `get rated(latest)` function returns the same result for all profiles.

Adding Activities in REST using a Java Client: Example

```
VisitorsClient client = new VisitorsClient(..);
Map<String,String> acts = new HashMap<>();
acts.put("product", " dell v560"); // type,data
acts.put("catalog", " laptops");
acts.put("web_page", "http://online.store.com/dell/v560");
client.addActivity("c11e123e-7fd0-4e55-9f6f-ef581e30d86a", acts);
```

Getting Activities in REST using a Java Client: Example

```
can get activities using two types:
getRated: fetches most rated activities
getLatest: fetches latest activities

getRated Method:
VisitorsClient client = new VisitorsClient(..);
List<Activity> res = client.getRated("c11e123e-7fd0-4e55-9f6f-ef581e30d86a",
"product", 5);

getLatest Method:
VisitorsClient client = new VisitorsClient(..);
List <Activity>res = client.getLatest("c11e123e-7fd0-4e55-9f6f-ef581e30d86a",
```

```
"product", 5);
```

```
5 - count. How many activities are needed
```

Adding Extended Attributes in REST using a Java Client: Example

```
VisitorsClient client = new VisitorsClient(..);
Map<String,String> attrs = new HashMap<>();
attrs.put("language", "en"); // name,value
attrs.put("gender", "male");
client. saveExtendedAttributes ("c11e123e-7fd0-4e55-9f6f-ef581e30d86a", attrs);
```

53.7.4.2 How to Use Extended Attributes and Activities in Visitor Services

The extended attributes are available for aggregation and enrichment in Visitor Services:

- Aggregation: Linked profiles have an additional profile named *extended* that contains all the extended attributes for the visitor.

- Enrichment: You can add enrichment rules for extended attributes as Extended Attribute Configuration's value. Visitor Services will use these enrichment rules to enrich visitors profiles. An example of extended attribute is a phone number that a visitor leaves for callback on a website. This number can be used as the visitor's extended attribute in Visitor Services to find the visitor's profiles from other profile stores. So the enrichment rules for this example would be `callback_phone=LDAP:Personal_Phone`.

The following process flow involves a sample online store, a visitor, and the Visitor Services activities functionality:

1. A visitor opens a browser and navigates to `http://online.store.com/dell/v560` and then selects the Dell v560 laptop.

2. The browser sends a request to the web server for the page with URL: `http://online.store.com/dell/v560`

3. The web server calculates the required data and performs the following operations:

   - (Optional) Fetches data from the database.

   - (Optional) Requests Visitor Services for the visitor's interests so they can include advertisements about the most relevant/interested (rated) goods.

     Sample code: `List<Activity> res = client.getRated("c11e123e-7fd0-4e55-9f6f-ef581e30d86a", "product", 5);`

     Visitor Services returns that this visitor is interested in Lenovo laptops (this information is based on previous activities).

   - Includes advertisements related to Lenovo laptops.

   - (Optional) Saves the recent activities of the visitor and requests Visitor Services to add activities:

     First: `visited page = 'http://online.store.com/dell/v560'`

     Second: `product = 'dell v560'`

Sample Code:

```
Map<String,String> acts = new HashMap<>();
acts.put("product", " dell v560'");
acts.put("web_page", "http://online.store.com/dell/v560");
client.addActivity("c11e123e-7fd0-4e55-9f6f-ef581e30d86a", acts);
```

53.8 Visitor Services Reference

Here is information about Visitor Services architecture, data model, aggregation templates, and providers such as Identity, Profile, Access.

- About the Visitor Services Architecture

- Identity Provider Reference

- Access Provider Reference

- Profile Provider Reference

- Aggregation Template Reference

- Diagnostics

- About the Visitor Services Data Model

- Glossary

53.8.1 About the Visitor Services Architecture

Visitor Services fetches visitor's profiles from different profile storages and lets developers store visitor profile information (such as city, education, interests, job profile, and so on) in a repository as different profile attributes. Visitor Services allows developers to query, enrich, store, and link visitor profiles for tracking usage and effective content targeting.

Visitor Services communicates with profile storages through profile providers that collect user data requested by applications. To access user information, Visitor Services relies on customers' single sign-on (SSO) solutions. Visitor Services uses identity providers to communicate with SSO solutions.

The following figure shows the Visitor Services architecture. Visitor Services includes the following integration layers:

- **Profile Providers**: Enable communication between Visitor Services and profile storages.

- **Identity Provider**: Enables communication between Visitor Services and its SSO solution.

- **REST Services**: Enable communication between Visitor Services and its client APIs.

Figure 53-16 Visitor Services Architecture
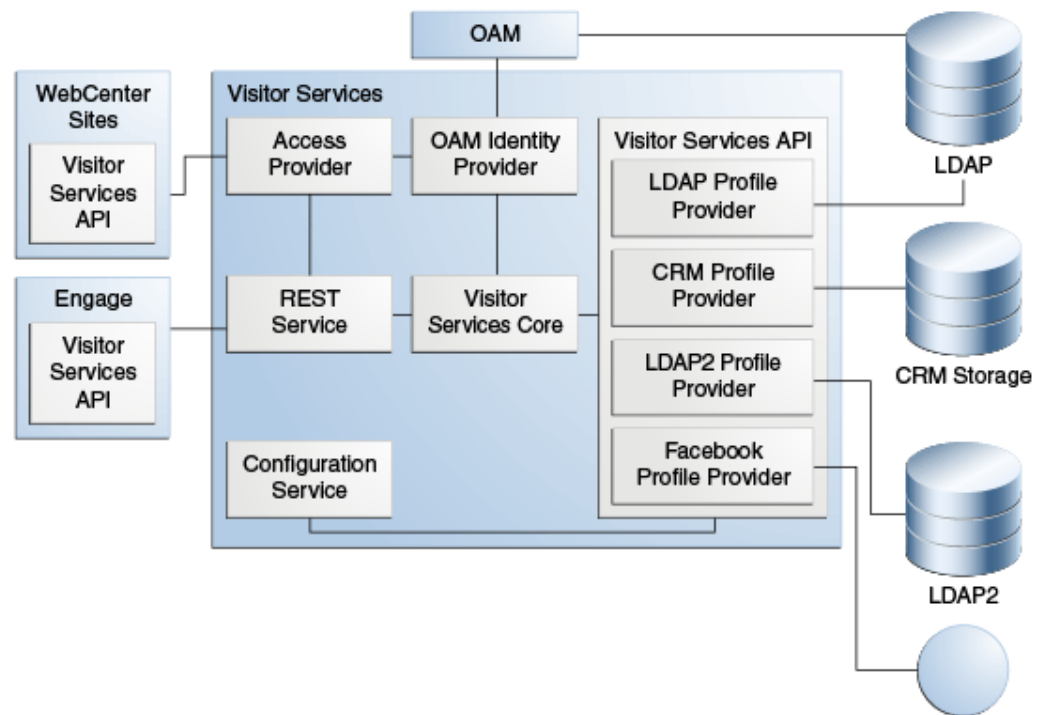
These integration layers are designed to support custom identity and profile providers in Visitor Services.

53.8.2 Identity Provider Reference

This reference includes the following topics:

- About the Identity Providers

- How Visitor Services Identifies Visitors to Your Website

53.8.2.1 About the Identity Providers

The Identity Provider Service enables integration between Visitor Services and the customer's SSO system to provide a communication mechanism between Visitor Services and the SSO system. When a browser requests the REST service for a visitor's profile, the REST service asks for an identify object from the identity provider. The identity provider passes on the identity object (consisting of DN and Provider) to the profile provider. If the user exists in the storage, the profile provider returns a unique Id. Otherwise, the user is considered a guest user, and therefore, a new unique Id is returned. Then, the REST service returns the visitor Id to the browser.

Using some identifiers such as a browser cookie ticket, identity provider provides the following information (called *identity* in Visitor Services) about the requested visitor:

- Visitor's login name (unique name in the storage).

- The name of the storage in which the visitor information is stored.

The identity provider functionality is an integral part of the Visitor Services application and cannot be accessed directly. Therefore, all requests about getting visitors' IDs are sent to the identity provider through REST services. Some SSO solutions work with external IDs such as cookie tickets, while others take HTTP

request as an input parameter. Keep in mind that the identity provider implementation should support both situations.

53.8.2.2 How Visitor Services Identifies Visitors to Your Website

Visitor Services helps identify website visitors by providing each with a unique visitor ID. It identifies visitors as one of the following types:

- **Authenticated**: Visitor Services identifies visitors who are registered with the website and can authenticate themselves through an SSO solution such as Oracle Access Manager (OAM) or SiteMinder.

- **Unauthenticated/Anonymous**: This type of visitor never signs in, so Visitor Services has no attributes for this user. While Visitor Services cannot identify anonymous visitors, if a client application (for example, WebCenter Sites) decides to register them explicitly, these visitors become authenticated visitors after their registration. The client applications can save attributes on their behalf post registration. The website can store the Visitor Services-generated visitor ID in a cookie and use it for associating the user sessions.

To handle authenticated visitor profiles, Visitor Services integrates with Oracle Access Manager (OAM) out of the box, including OAM Mobile and Social. For other SSO solutions, developers can create extensions and configure custom identity providers.

See Configuring an Identity Provider.

53.8.3 Access Provider Reference

This service is used to protect the connection between Visitor Services and applications. You can configure either basic, LDAP-based authentication, or any other, to verify the authenticity of the clients that interact with Visitor Services. If authentication is not configured through the access provider service, all resources are considered public and can be accessed by anyone. It is strongly recommended that you set up the access provider in production systems.

> **Note:**
>
> The JavaScript API used to connect applications with Visitor Services on the client side is considered a public API, and it is not supported by the access provider service.

This reference includes the following topics:

- About Container Protection and Visitor Services Protection

- How Container Protection Works

- How Visitor Services Protection Works

53.8.3.1 About Container Protection and Visitor Services Protection

The access provider service provides the following types of protection:

- Container Protection: All the information about users is stored in the container storage, and Visitor Services only checks for users' roles. Oracle recommends using the container protection if client's web server allows to configure basic authentication. The basic authentication requires the following:

- Web Server: `user = "username", password="pass", role="svsclient"`

- Access Provider: Upload the `access-provider-container-basic.jar` bundle. Config: `role= svsclient`

- REST Client: Initialize the client with `new HttpAuthentication("username "," pass ")`

- Visitor Services Protection: Information about users can be stored in LDAP, database, etc. If client's web server doesn't support the basic authentication mechanism, use the Visitor Services protection.

   The basic LDAP protection from Visitor Services requires the following:

   - Access Provider: Upload the `access-provider-ldap-basic.jar` bundle.

     Config:

     * LDAP connection settings are the same as LDAP profile provider.

     * auth type settings: `HeaderName=Visitors-Authorization` and `AuthAlias=Visitors-Basic`

   - LDAP with user's `username` along with `password`.

   - REST Client: Initialize the client with `new VisitorsHttpAuthentication("username","pass")`

53.8.3.2 How Container Protection Works

1. Client sends a profile information request to Visitor Services. This request contains the required authentication, that is, auth token in the Authentication header.

2. Web server halts this request and checks the Authentication header.

3. If the user cannot be authenticated, the web server returns the 401 error.

4. If the user is authenticated successfully, the web server adds user roles to the request.

5. The request comes to the access provider.

6. The function `boolean check(HttpServletRequest request)` checks the request for the required role.

7. If the role check is successful, Visitor Services processes the request.

53.8.3.3 How Visitor Services Protection Works

1. Client sends a profile information request to Visitor Services. This request contains the required authentication information.

2. When the request comes to the access provider service, the function `boolean check(HttpServletRequest request)` checks the request for the authentication information.

3. The access provider checks its storage to verify the authentication information that came with the request.

4. Visitor Services processes the request.

53.8.4 Profile Provider Reference

This reference includes the following topics:

- About the Profile Providers and Enrichment Service
- How Visitor Services Gathers and Enriches Visitor Attributes from Multiple Channels

53.8.4.1 About the Profile Providers and Enrichment Service

Profile providers let Visitor Services fetch visitor profile information from respective repositories. The Enrichment service let Visitor Services fetch a visitor's profile information from all the repositories for which profile providers exist.

- Profile Providers
- Enrichment Service
- How Profile Provider and Enrichment Services Work

53.8.4.1.1 Profile Providers

The main task of Visitor Services is to retrieve visitor information from external storages. These storages could be text files containing some structured data, an LDAP, or external web services built on some database. To fetch visitor data from profile storages, developers can implement custom profile providers and plug them in to Visitor Services. For each profile storage, there can be one profile provider.

For customers who use LDAP and Eloqua, profile providers are available out of the box. These profile providers are available as OSGi bundles that are installed using the Admin interface. These OSGi bundles are actual JAVA-based implementation of the communication required with the desired storage.

> **Note:**
>
> To use Eloqua profile provider and get the visitor Id, the –
> `DUseSunHttpHandler=true` parameter must be set in the Visitor Services
> managed server if the application server being used is the WebLogic server.

53.8.4.1.2 Enrichment Service

The Enrichment service enables Visitor Services to search for visitor information in different profile storages using profile providers. Each profile provider is configured with a single set of rules that enable Visitor Services to search for information that all profile providers search for. Therefore, enrichment rules collect visitor information from all profile providers. These rules have the following structure:
`AttributeNameInCurrentProfileProvider=DifferentProfileProviderName:AttributeNameInThatProfileProvider`.

When Visitor Services receives a request to get a visitor's profile from LDAP, in the background, Visitor Services also searches for this visitor's profile in CRM by enrichment rules specified for the corresponding profile. If it finds the visitor profile in CRM, it searches Eloqua using email address as the search criterion. Enrichment process is time consuming. Therefore, when a visitor's profile is retrieved from a

storage using profile provider for the first time, the process runs as a separate flow in the background.

When Visitor Services has collected visitor information from multiple storages, it compiles them (based on aggregation templates) for applications such as Engage.

53.8.4.1.3 How Profile Provider and Enrichment Services Work

The figure below shows how Visitor Services communicates with profile providers to get raw profiles from profile storages and apply enrichment rules to the profiles.

Figure 53-17   Process Flow between Visitor Services and Profile Providers
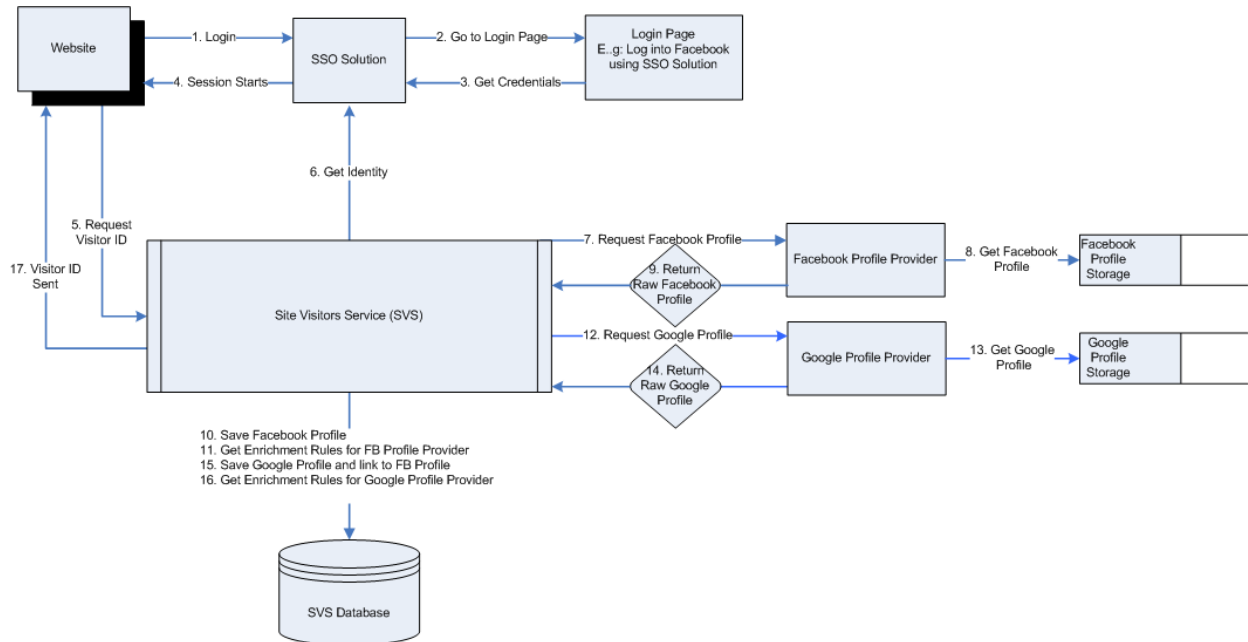
Procedure based on the Process Flow Depicted in the figure above:

1. A visitor tries to log into Facebook.

2. The login request is passed via the SSO solution to the Facebook login page.

3. Visitor's credentials are passed to the SSO solution.

4. The session starts.

5. The website requests Visitor Services for visitor ID.

6. The SSO solution passes visitor's identity to Visitor Services.

7. Visitor Services requests the Facebook profile provider for visitor's Facebook profile.

8. The Facebook profile provider gets the visitor's Facebook profile from the Facebook profile storage.

9. The Facebook profile provider returns the raw Facebook profile to Visitor Services.

10. The raw Facebook profile is saved to Visitor Services database.

11. Visitor Services gets enrichment rules for Facebook profile provider.

12. Visitor Services requests Google profile provider for Google profile.

13. The Google profile provider gets the visitor's Google profile from the Google profile storage.

14. The Google profile provider returns the raw Google profile to Visitor Services.

15. Google profile and Facebook profile are linked and saved in Visitor Services database.

16. Visitor Services gets enrichment rules for Google profile provider.

17. Visitor Services sends the visitor ID to the website.

53.8.4.2 How Visitor Services Gathers and Enriches Visitor Attributes from Multiple Channels

Using **profile providers**, Visitor Services searches for and fetches visitor data from multiple profile stores such as LDAP, Facebook, Eloqua, and CRM. Profile providers for LDAP and Eloqua are provided. For other profile stores, developers can create and configure custom profile providers.

For each visitor a profile provider finds, Visitor Services creates a **raw visitor profile**, with the **visitor ID** and attributes found. For example, a visitor's name and email may come from an LDAP profile provider, and his or her shipping address may come from a CRM provider.

To help build a comprehensive visitor view, Visitor Services can search for and gather visitor attributes from profile providers through **profile enrichment**, and link visitor profiles. For example, for a visitor who logged in through OAM, Visitor Services might find an email address in an LDAP provider profile and search for a matching email address in an Eloqua profile.

See Configuring One or More Profile Providers.

53.8.5 Aggregation Template Reference

This reference includes the following topics:

- About Aggregation Templates

- How Visitor Services Merges Raw Visitor Profiles into a Single Aggregated Profile

- How Visitor Services Makes Aggregated Visitor Profiles Available for Targeting, Testing, and Analysis

53.8.5.1 About Aggregation Templates

An aggregation template is used to aggregate profile attributes from one or more profile providers defined in Visitor Services. For example, different profile storages use different names for the same attributes. A visitor's name and email Id may be different in Google+ and Facebook. The Visitor Services aggregation service provides aggregation templates that contain the following parameters to identify the attributes of the same types irrespective of how they are named across different profiles:

- Metadata of the profile (List of attributes)

- Set of rules to combine information from different profiles

The requests made to the aggregation service go through the Visitor Services API to which clients send visitor IDs and the aggregation templates. Each client has its own profile structure irrespective of the visitor profile currently available in Visitor Services. The aggregation service builds a profile using the application's template. An aggregation template is a set of required attributes and rules using which attributes are collected and compiled from different raw profiles. The structure of the aggregated profile is always the same regardless of the list of available profiles in Visitor Services for that visitor and regardless of the structure of raw profiles from different profile providers.

53.8.5.1.1 How the Aggregation Service Works

The figure below shows how the aggregation service gets all linked profiles and the required template when requested.

Figure 53-18   Process Flow: How the Aggregation Service Works
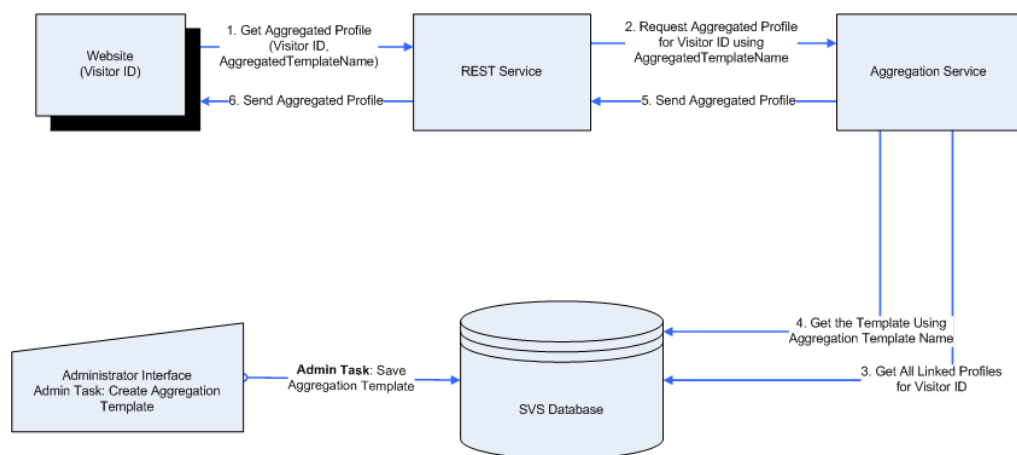
Procedure based on the Process Flow Depicted in Figure 53-18:

1. To get an aggregated profile of a visitor with the specific attributes, website provides the visitor ID and the aggregation template name to the REST service.

2. The REST service sends the visitor ID and aggregation template name to the aggregation service.

3. The aggregation service gets all the linked profiles available for the visitor ID from the Visitor Services database.

4. The aggregation service gets the aggregation template for the aggregation template name provided by the REST service.

5. The aggregation service sends the aggregated profile based on the aggregated template to the REST service.

6. The REST service sends the aggregated profile to the website.

53.8.5.2 How Visitor Services Merges Raw Visitor Profiles into a Single Aggregated Profile

Profile aggregation uses aggregation templates to consolidate visitor attributes from multiple visitor profiles into an aggregated profile. (Raw visitor profiles remain separate.) You can create different aggregation templates depending on business need. For example, the marketing department might request one template that provides name, email address from LDAP, and phone number from CRM, and another

template that provides name from LDAP and mailing address data from CRM profiles.

In addition to combining visitor profiles from different sources, you can also use an aggregation template to use stored profile information, compute a value found (for example, total orders placed during the past year), and add the computed value as a profile attribute.

See Creating One or More Aggregation Templates.

53.8.5.3 How Visitor Services Makes Aggregated Visitor Profiles Available for Targeting, Testing, and Analysis

Visitor Services collects, links, and stores attributes into aggregated visitor profiles. To make use of the profiles, WebCenter Sites components must request them via Visitor Services APIs (Java Client and JavaScript) during the runtime.

Marketers can use visitor profiles from Visitor Services to determine and create Engage segments. For example, a marketer might use visitor profile attributes from Visitor Services such as age and income to create segments and deliver recommendations.

For unauthenticated and unknown visitors, marketers can create segments for them based on their attributes such as device used to access the site, time zone, locale, browser, referrer (Facebook, Twitter, Bing), or IP address.

53.8.6 Diagnostics

Cache Tool Resources

If you would like to see what's inside Visitor Services, you should use the debug tool.

Cache Tool Resources:

GET

```
http://<host>:<port>/<context>/rest/v1/cachetool/{region}/list
```

The resource endpoint can be used to fetch the list of all the providers currently installed in Visitor Services.

REQUEST:

Path Parameters: `Name Description Format`

`Region:` Accepts the values `common` and `shared_cache`. `Common` returns the list of all the installed providers, and `Shared_cache` returns the list of all the installed providers with the date of install/update `String`.

RESPONSE:

Supported Media Types: `application/json`

200 Response: The list of installed providers.

Example 53-1   Example 1: Fetch the list of currently installed providers

```
curl -i -H "Accept: application/json" -X GET
http://<host>:<port>/<context>/rest/v1/cachetool/common/list
Response:
Content-Length:1111
Content-Type:application/json

{
```

```
    "type": "cacheToolResponse",
    "status": "success",
    "entry": {
      "entry": [
{
          "key": "profileProviderConfig.pr1",
          "value": "profileProviderConfig.pr1"
        },
        {
          "key": "profileProviderConfig.pr2",
          "value": "profileProviderConfig.pr2"
        },
        {
          "key": "profileProvider.pr1",
          "value": "profileProvider.pr1"
        },
        {
          "key": "profileProvider.pr2",
          "value": "profileProvider.pr2"
        },
        {
          "key": "identityProviderConfig.identityProvider1",
          "value": "identityProviderConfig.identityProvider1"
        },
        {
          "key": "identityProvider.identityProvider1",
          "value": "identityProvider.identityProvider1"
        }
      ]
    }
}
```

Example 53-2   Example 2: Fetch the installed/updated time of currently installed providers

```
curl -i -H "Accept: application/json" -X GET
http://<host>:<port>/<context>/<rest>v1/cachetool/shared_cache/tool
Response:

Content-Type:application/json
 {
   "type": "cacheToolResponse",
   "status": "success",
   "entry": {
     "entry": [
       {
         "key": "profileProviderConfig.pr1",
         "value": "Fri Sep 04 12:03:41 IST 2015"
       },
       {
         "key": "profileProviderConfig.pr2",
         "value": "Fri Sep 04 12:03:41 IST 2015"
       },
       {
         "key": "profileProvider.pr1",
         "value": "Fri Sep 04 12:03:41 IST 2015"
       },
       {
         "key": "profileProvider.pr2",
         "value": "Fri Sep 04 12:03:41 IST 2015"
       },
```

```
        {
          "key": "identityProviderConfig.identityProvider1",
          "value": "Fri Sep 04 12:03:41 IST 2015"
        },
        {
          "key": "identityProvider.identityProvider1",
          "value": "Fri Sep 04 12:03:41 IST 2015"
        }
      ]
    }
}
```

53.8.7 About the Visitor Services Data Model

The configurations of identity, profile, and access providers are stored in the
WebCenter Sites database and retrieved by Visitor Services through the Sites REST
API. Visitors' raw profiles, attributes, and activities are stored in the local Visitor
Services database.

WebCenter Sites database includes the tables described in the table below. These
tables are created during WebCenter Sites installation.

Table 53-4   Tables in WebCenter Sites Database

Table Name	Description
WCS_ProfileProviders	Contains profile providers' implementation, configuration, and enrichment rules.
WCS_IdentityProviders	Contains identity providers' implementation and configuration.
WCS_AccessPrividers	Contains access Providers' implementation and configuration.
WCS_AggregatedTemplat es	Contains aggregation templates.
WCS_VisitorsConfig	Contains Visitor Services general configuration.

The local Visitor Services database includes the tables described in the table below.

Table 53-5   Tables in the Local Visitor Services Database

Table Name	Description
WCS_VIS_Profiles	Contains raw profiles from profile storages, provided by profile providers.
WCS_VIS_Attributes	Contains activities registered by applications.
WCS_VIS_Activity	Contains extended attributes saved by applications.

The figure below shows the Visitor Services data model and the tables each of the
databases includes.

Figure 53-19   Visitor Services Data Model
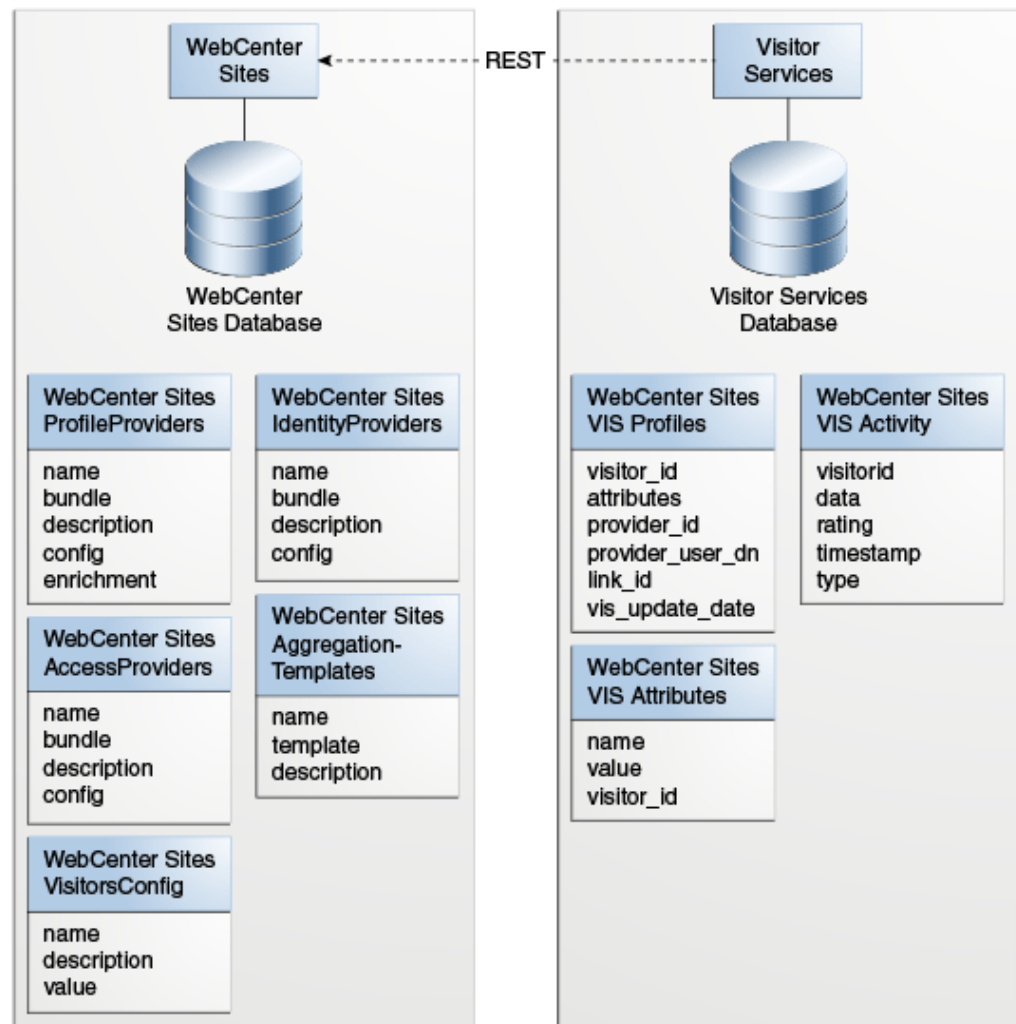
53.8.8 Glossary

Aggregation

The process of retrieving an aggregated profile from the raw profiles based on the aggregation rules defined in Visitor Services.

Aggregated Profile

Unique profile attributes compiled by Visitor Services from different linked profiles of a visitor.

Aggregation Template

Configurations to retrieve profile attributes from raw profiles to create an aggregated profile.

Enrichment

The process of capturing raw profiles of a visitor from all the profile storages based on enrichment rules defined in Visitor Services.

External Id

An external user identifier provided by a SSO solution.

Identity

A pair of values: visitor's profile provider name and distinguished name (DN). This pair comes from the identity provider.

Identity Provider

An entity implemented by developers which accepts an externalID to provide the profile provider name and user DN pair from the corresponding identity store to Visitor Services.

Identity Store

A storage that contains visitors' information. It could be an LDAP or a database.

Linked Profile

Data provided by Visitor Services that contains all linked raw profiles in the form of a few lines in the Visitor Services database.

Profile Provider

An entity implemented by developers to receive visitors' profiles from specific profile stores. In general, one provider is implemented per profile store.

Profile Storages

External storages that contain information about visitors' profiles.

Raw Profile

Visitor data provided by the profile provider from its profile store.

Stored Profile

An internal Visitor Services entity that contains processed profile data in the form of RawProfile, Identity, LinkId, and VisitorId.

SSO Solution

An authentication system which provides the single sign-on functionality between Visitor Services and applications. Visitor Services comes packaged with the OAM SSO system.

Visitor Services Database

The local database used by Visitor Services to store visitors' profile data and settings such as aggregation rules and mapping rules.

Part XIV

Enabling Convergence between Eloqua and Oracle WebCenter Sites

WebCenter Sites integrates tightly with marketing automation tools like Eloqua to provide reusable content for campaigns and blogs. To unify the customer experience and consistency across channels, marketers can share and reuse the content from WebCenter Sites and deliver higher conversion rates.

- Integrating Eloqua Cloud Marketing Service with Oracle WebCenter Sites

54

Integrating Eloqua Cloud Marketing Service with Oracle WebCenter Sites

WebCenter Sites can use Eloqua visitor profile data to target specific content to the visitor by personalizing their visit to all those pages where they navigate around on your website. By sharing both content and visitor profile data across Eloqua and WebCenter Sites, the visitor gets a consistent personalized experience that leads to higher conversions. Read further to find out how you can integrate the Eloqua Cloud Marketing Service (Eloqua) with Oracle WebCenter Sites.

See these topics:

- Eloqua Integration Overview
- Accessing WebCenter Sites Content in Eloqua
- Accessing Eloqua Forms in WebCenter Sites

54.1 Eloqua Integration Overview

Integrating Eloqua with WebCenter Sites provides marketers with the ability to converge the visitor's online experience with their marketing campaigns, as well as more effectively manage the visitor journey. The integration eliminates content duplication and streamlines content targeting across channels.

This is achieved through the following functionality enabled by the integration:

- **Embedding WebCenter Sites content into Eloqua forms and landing pages.** Drag-and-drop placement of existing WebCenter Sites content ensures consistency of the marketing message and a seamless visitor experience across channels. Specific content can be targeted at specific visitors or campaign contacts.

- **Embedding Eloqua forms into WebCenter Sites pages.** Drag-and-drop placement of Eloqua forms into WebCenter Sites pages eliminates the involvement of the site administrator or developer in the lead generation process.

Existing visitor segment and contact data can be automatically passed from Eloqua to WebCenter Sites for targeted content delivery.

54.1.1 Integrating Cloud Marketing Services with WebCenter Sites

Eloqua uses its AppCloud framework to communicate with the WebCenter Sites content service running on the target WebCenter Sites instance over secure HTTP authenticated via Oauth 1.0. Supported operations include get, search, list, copy, modify, and delete. The secure HTTP calls passed between the WebCenter Sites content service and the Eloqua AppCloud framework can be logged for troubleshooting purposes.
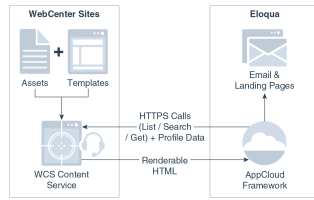
The WebCenter Sites administrator decides which asset types are available to Eloqua marketers. Content from WebCenter Sites is delivered to Eloqua as renderable HTML code embedded directly into the target e-mail or landing page.

To target specific content at specific visitors or campaign contacts, the template that renders the desired WebCenter Sites content can include placeholder tags that automatically accept visitor segment and contact field data from Eloqua and substitute it during rendering.

Eloqua forms are placed into WebCenter Sites pages by reference as proxy assets and rendered by WebCenter Sites templates. The Eloqua administrator decides which Eloqua forms are available to the WebCenter Sites instance.
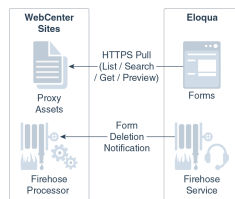
If a form that has been embedded into a WebCenter Sites page is deleted, Eloqua's Firehose service notifies the WebCenter Sites administrator, or another individual or group, via e-mail so that the affected page(s) can be updated accordingly. The notification message lists the affected pages.

WebCenter Sites ships with sample rendering templates to get you started with the integration process.

54.1.2 Accessing Eloqua Documentation

Some procedures in this guide call for completing pre-requisite steps. For information on completing these steps, consult the Eloqua documentation available at:

http://docs.oracle.com/cloud/latest/marketingcs_gs/OMCAA/index.html

54.2 Accessing WebCenter Sites Content in Eloqua

To embed WebCenter Sites content in Eloqua e-mails and landing pages, you need to configure your Eloqua instance with the WebCenter Sites content service, ensure that only the authorized Eloqua instance can access content within the target WebCenter Sites instance, and determine which WebCenter Sites asset types and associated templates are available to Eloqua marketers.

See these topics:

- Configuring Eloqua for Access to the WebCenter Sites Content Service

- Configuring OAuth Authentication

- Making WebCenter Sites Assets Available within Eloqua

54.2.1 Configuring Eloqua for Access to the WebCenter Sites Content Service

Before marketers can embed existing WebCenter Sites content into Eloqua e-mails and landing pages, you must configure your Eloqua instance to communicate with the WebCenter Sites content service. The example URLs below assume the default servlet path of `/cs/ContentServer` - adjust your URLs as appropriate for your environment. Also, if you have not already done so, create a new provider in Eloqua to separate the WebCenter Sites content service from other services and applications.

> **Note:**
>
> Your WebCenter Sites instance must be configured to accept HTTPS connections. Only ports 443 and 8443 are supported. All URLs, property names, and their values are case-sensitive.

To configure Eloqua for accessing the WebCenter Sites Content Service, you must do the following:

1. Log into the target Eloqua site as its administrator.

2. In the top bar, click **Settings** and select **AppCloud Developer**.

3. Click **Create New App** and fill in the form that appears as follows:

Table 54-1   Create New App Fields

Field	Value
Icon URL	`https://wcs-instance.company.com/cs/images/eloqua/ AppIcon.png`
Enable URL	`https://wcs-instance.company.com/cs/ContentServer? pagename= EloquaIntegration/ Enable&installId={InstallId}&appId={AppId}& callbackUrl={CallbackUrl}`
Status URL	`https://wcs-instance.company.com/cs/ContentServer? pagename= EloquaIntegration/SitesStatus`
Callback URL	`https://wcs-instance.company.com/cs/ContentServer? pagename= EloquaIntegration/Enable`

4. Click **Save App** to save your changes.

5. In the **App Details** page that appears, click **Show Secret** in the **OAuth** section, copy the client secret value displayed in the pop-up, and save it to a text file. (You need this value to configure the authentication of Eloqua requests within WebCenter Sites later in this guide.)

6. In the **Services** section click **Add Service**, and then click **Content** in the pop-up that appears.

7. Fill in the fields in the **Content Service Details** form as follows:

Table 54-2   Content Service Details Fields

Field	Value
Icon URL	https://*wcs-instance.company.com*/cs/images/ eloqua/ ContentServiceIcon.png
Create URL	https://*wcs-instance.company.com*/cs/ ContentServer?pagename= EloquaIntegration/ Instances/Add&instanceId={InstanceId}
Configure URL	https://*wcs-instance.company.com*/cs/ ContentServer?pagename= EloquaIntegration/ ContentSearch&instanceId={InstanceId}& installId={InstallId}
Delete URL	https://*wcs-instance.company.com*/cs/ ContentServer?pagename= EloquaIntegration/ Instances/Delete&instanceId={InstanceId}
Copy URL	https://*wcs-instance.company.com*/cs/ ContentServer?pagename= EloquaIntegration/ Instances/Copy&instanceId={InstanceId}& originalInstanceId={OriginalInstanceId}
Layout	**Flow** Flow layout mode allows the embedded content to be dynamically resized.
Max-records	The maximum number of rendered assets passed from WebCenter Sites to Eloqua in a single call. Oracle recommends a value below 5000; however, if you experience issues with passing very large pieces of content, lower this value. For more information, consult Eloqua documentation.
Content-type	**application/json**
Landing Page Notification URL	(Optional) To make WebCenter Sites content available in Eloqua landing pages: https://*wcs-instance.company.com*/cs/ ContentServer?pagename= EloquaIntegration/ Processor/ LandingPageNotification&instanceId= {InstanceId}&installId={InstallId}&execution Id={ExecutionId}
E-mail Page Notification URL	(Optional) To make WebCenter Sites content available in Eloqua e-mails: https://*wcs-instance.company.com*/cs/ ContentServer?pagename= EloquaIntegration/ Processor/EmailNotification&instanceId= {InstanceId}

Fill in the remaining fields as appropriate for your environment, then click **Save** to commit your changes.

8. Save your changes and return to the WebCenter Sites content service properties page.

9. Click the link in the Catalog section to add the WebCenter Sites content service to your cloud service catalog. When prompted, click **Accept**.

10. Install the WebCenter Sites cloud service into your Eloqua instance.

This makes it available within the Eloqua marketer interface for the target Eloqua instance only.

a. In the Eloqua interface, click **Settings**, and then select **Setup**.

b. Click **AppCloud Catalog**.

c. Select the **Catalog** tab to display the list of available (but not installed) services.

d. Locate your target application and click **Install**.

> **Note:** To maintain security within your Eloqua instance, do not publish the WebCenter Sites content service.

54.2.2 Configuring OAuth Authentication

Secure HTTP calls from Eloqua to the WebCenter Sites content service are authenticated via OAuth 1.0 to ensure that only the authorized Eloqua instance can access content within the target WebCenter Sites instance.

To facilitate this authentication process, you must provide the Eloqua client secret value to the WebCenter Sites content service as follows:

1. Open the Property Management Tool and in the **Search by** section, do one of the following:

- In the **Name** field, enter the name of the property you want to modify.

- In the **Category** drop-down menu, select `EloquaIntegration`.

2. Click **Search**.

3. In the **Key** column, select the name of the property.

4. In the **Value** field, enter the new value.

5. Click **Save**.

6. Repeat steps 2 through 6 for all the properties you want to change.

7. Stop and restart the application server to apply the changes.

54.2.3 Making WebCenter Sites Assets Available within Eloqua

You need to decide which WebCenter Sites asset types and associated templates should be available to Eloqua marketers. Rendering templates must be created specifically for rendering WebCenter Sites content within Eloqua e-mails and landing pages according to the best practices listed in the next section.

54.2.3.1 Creating Asset Rendering Templates for Eloqua

WebCenter Sites content is embedded into Eloqua e-mails and landing pages not by reference, but as fully renderable HTML code. Because of this, keep in mind the following guidelines when creating templates for rendering WebCenter Sites assets within Eloqua e-mails and landing pages:

- Use absolute URLs and paths,

> **Note:** The `render:getbloburl` tag does not generate an absolute URL to the target blob unless you explicitly pass the `scheme` and `authority` values. For more information on these values, see the *Oracle Fusion Middleware Tag Reference for Oracle WebCenter Sites Reference.*

- Embed all styling (CSS) within the template,

- The element used in the template must be configured as follows:

    Type: Layout
    Usage: Element defines a whole HTML page and can be called externally.

When coding a template to render assets containing custom contact data pulled from Eloqua contact fields, you must add a `{{C_fieldname}}` placeholder within the template code for each custom field (where `C_fieldname` is the database record name of the target Eloqua field), then complete the steps in the next section. Once configured, the contact field values are string-substituted in the returned HTML content.

When rendering customized content, WebCenter Sites returns a JSON array containing the customized rendered HTML code, one object per each customized asset. The number of rendered assets passed within this array for each call is governed by the `max-records` setting (configured when setting up the WebCenter Sites content service within Eloqua).

54.2.3.2 Configuring Asset Type Visibility in Eloqua

To make assets of the desired types visible to Eloqua marketers, you must do the following:

1. Decide which asset types you want to make visible to the Eloqua marketers.

2. Decide on a friendly name for each asset type so that Eloqua marketers can easily identify the type of content it represents and add them as a comma-separated list to the value of the `eloquacloudapp.assettype.friendlyname` property through the Property Management Tool. For example:

```
eloquacloudapp.assettype.friendlyname=Business Article,Sports
Article,

Travel Brochure
```

3. Associate the friendly names with their corresponding WebCenter Sites asset type names. For each consecutive friendly name, add the following property through the Property Management Tool:

```
eloquacloudapp.assettype.N=AssetTypeName:AssetSubTypeName
```

where:

- *N* is consecutive position of the corresponding friendly name in the comma-separated list in the property `eloquacloudapp.assettype.friendlyname`, starting with 0.

- AssetTypeName is name of the target WebCenter Sites asset type.

- AssetSubTypeName(Optional) is name of the target asset sub-type.

For example:

```
eloquacloudapp.assettype.0=Article:BusinessArticle
```

```
eloquacloudapp.assettype.1=Article:SportsArticle
```

```
eloquacloudapp.assettype.2=TravelBrochure
```

4. Specify the template(s) that you have developed to render assets of each exposed type and sub-type. For each consecutive friendly name (and thus the corresponding asset type and sub-type), add the following property through the Property Management Tool:

```
eloquacloudapp.templates N=TemplateName:SiteName.
```

where:

- *N* is consecutive position of the corresponding friendly name in the comma-separated list in the property `eloquacloudapp.assettype.friendlyname`, starting with `0`.

- TemplateName is name of the corresponding template for the target asset type and sub-type. When using a typeless template, you must precede its name with a `/` (forward slash).

- SiteName is name of the WebCenter Sites site in which both the assets of the target asset type and sub-type, as well as the corresponding templates reside. Do not specify a template that resides in a site other than that the site holding the assets of the target type and sub-type, as this results in missing content.

To specify multiple templates for each target asset type, append additional *TemplateName*:*SiteName* values to the property, separated by commas.

For example:

```
eloquacloudapp.templates.0=BusinessArticleLayout1:mySite,
BusinessArticleLayout2:mySite
```

```
eloquacloudapp.templates.1=SportsArticleLayout1:mySite,
SportsArticleLayout2:mySite
```

```
eloquacloudapp.templates.2=TravelBrochureLayout:mySite
```

5. (Optional) Specify the Eloqua contact fields used by the templates configured in step 4 using their database record names. For each corresponding `eloquacloudapp.templates.`*N* entry, add the following entry through the Property Management Tool:

```
eloquacloudapp.templatefields.N=C_Field1|C_Field2,C_Field3|C_Field4
```

where:

- *N* is consecutive position of the corresponding friendly name in the comma-separated list in the property `eloquacloudapp.assettype.friendlyname`, starting with `0`.

- *Field1...Field4* is Database record names of the target contact fields. For each template specified in the corresponding `eloquacloudapp.templates.`*N* entry, add a comma-separated entry consisting of pipe-separated field names.

For example:

```
eloquacloudapp.templatefields.0=C_FirstName|C_LastName,
```

```
C_EmailAddress|C_PhoneNumber
```

> **Note:**
>
> You can determine the database record names for the target fields by
> navigating to **Settings**>**Setup**>**Fields & Views**>**Contact Fields** within the
> Eloqua interface. The names begin with `C_`.
>
> The placeholder in the template is `{{C_fieldname}}` and the fields are string-
> substituted in the returned HTML content.

6. Assign images that represent the type of content stored by assets of the exposed
   asset types and sub-types so that Eloqua marketers can more easily identify the
   content they want to use. The image is displayed in the asset selection dialog along
   with the asset's name and modification date.

   For each consecutive friendly name (and thus the corresponding asset type and
   sub-type), add the following image attribute definition entry through the Property
   Management Tool:

   ```
   eloquacloudapp.attributename.N=AttributeName1:AttributeName2
   ```

   where:

   - *N* is consecutive position of the corresponding friendly name in the comma-
     separated list in the property
     `eloquacloudapp.assettype.friendlyname`, starting with `0`.

   - AttributeName is name of the target attribute.

   > **Note:** The complexity of your attribute hierarchy might dictate specifying
   > several levels of attributes.

   For example:

   ```
   eloqua.CloudApp.attributename.0=articleImage:largeImage
   ```

   ```
   eloqua.CloudApp.attributename.1=sportsImage:smallImage
   ```

   ```
   eloqua.CloudApp.attributename.2=travelImage:color:thumbnail
   ```

54.2.4 Placing WebCenter Sites Content into Eloqua E-Mails and Landing Pages

The Eloqua marketer places WebCenter Sites content into an Eloqua e-mail or landing
page as follows.

To place WebCenter Sites content into Eloqua e-mails and landing pages, you must do
the following:

1. In the target Eloqua instance, create a new, or edit an existing e-mail or landing
   page.

2. In the content bar on the left, click **Cloud Content**.

3. In the **Cloud Content** pop-up dialog that appears, locate the WebCenter Sites
   content service and drag it into the e-mail or page; position and size the resulting
   content frame as necessary.

> **Note:** Note that if the page layout is set to `flow`, the width of this content frame defines the width of the content it holds.

4. Double-click the content frame you just placed to display the asset selection dialog and locate the content you want to embed; the drop-down menus allow you to browse through available asset types and sub-types.

5. Once you have located the desired asset, select it from the list, then do one of the following:

   • To preview the asset before embedding it, click **Preview** and select the desired rendering template from the drop-down list,

   • To embed the asset, click **Embed** and select the desired rendering template from the drop-down list,

   • To use another asset, locate and select it in the content selection dialog,

   > **Note:** The content does not immediately appear in the e-mail or page you are editing - you must preview the e-mail or page, or finalize it in order to view the embedded content

6. Complete editing your e-mail or page and save your changes.

54.3 Accessing Eloqua Forms in WebCenter Sites

The marketers will be able to access Eloqua forms after you've configured a connection between WebCenter Sites and the target Eloqua instance, enabled the EloquaForm proxy asset type, registered the Firehose service, and created the rendering template for Eloqua assets.

> **Note:**
>
> This topic assumes you have successfully completed the steps in Accessing WebCenter Sites Content in Eloqua.
>
> Your WebCenter Sites instance must be configured to accept HTTPS connections. Only ports 443 and 8443 are supported.
>
> All URLs, property names, and their values are case-sensitive.
>
> Eloqua communicates with TLS version 1. Therefore, add -Dweblogic.security.SSL.minimumProtocolVersion=TLSv1 in your weblogic settings to enable communications with eloqua.

See these topics:

• Specifying the Eloqua Instance Connection Parameters

• Enabling the EloquaForm Proxy Asset Type

• Registering the Eloqua Firehose Service

• Creating the Rendering Template for the EloquaForm Assets

54.3.1 Specifying the Eloqua Instance Connection Parameters

To allow WebCenter Sites to connect to the target Eloqua instance, add the following properties through the Property Management Tool:

Property	Value
`eloqua.company`	Name of the company for which the target Eloqua account has been established.
`eloqua.loginurl`	URL to login to the Eloqua account.
`eloqua.username`	Name of target Eloqua account.
`eloqua.password`	Password for the target Eloqua account.
`eloqua.proxyHost`	(Optional) If your organization uses an Internet proxy, specify the full host name of the proxy server.
`eloqua.proxyPort`	(Optional) If your organization uses an Internet proxy, specify the port on which the proxy server listens for connections.

54.3.2 Enabling the EloquaForm Proxy Asset Type

You must enable the `EloquaForm` asset type (installed with the patch) for the target WebCenter Sites site. For instructions, see the WebCenter Sites documentation.

54.3.3 Registering the Eloqua Firehose Service

The Eloqua Firehose service notifies you whenever a form that has been embedded into a WebCenter Sites page has been deleted from Eloqua. This minimizes the chance of delivering pages that are missing content.

1. Log in to the target Eloqua instance as the administrator.

2. In the top bar, click **Settings**, and then select **AppCloud Developer**.

3. In the list of services, click the WebCenter Sites content service you have created.

4. In the Services section click **Add Service**, and then select **Firehose** from the pop-up dialog.

5. In the form that appears, fill in the fields as follows:

Field	Value
Icon URL	`https://wcs-instance.company.com/cs/images/eloqua/AppIcon.png`
Name	WebCenter Sites Firehose
Description	WebCenter Sites Firehose Notification Service

Field	Value
Pattern	`Form.Deleted`
Notification URL	`https://wcs-instance.company.com/cs/ContentServer?pagename=EloquaIntegration/Processor/FirehoseNotification`

6. Click **Save** to store your changes.

7. Add the following line to the `protectedPages`list in the `web.xml` file on your WebCenter Sites instance:

   `EloquaIntegration/Processor/FirehoseNotification`

Once an Eloqua form has been embedded into a WebCenter Sites page, a rendered copy of that page, including the Eloqua form, is stored in the WebCenter Sites cache. If an Eloqua marketer deletes the embedded form within Eloqua, that cached page copy is no longer valid.

When notified by the Firehose service, the WebCenter Sites contributor responsible for the affected page must manually remove the deleted Eloqua form from the page and republish it; otherwise, the page is delivered to the visitor's browser incomplete.

54.3.4 Configuring the Form Deletion Notification

To define the body, content, recipient address, and other parameters of the form deletion notification message sent by the Firehose service, add the following properties through the Property Management Tool:

Property	Value
`eloquanotification.email.address`	E-mail address that receives the notification message.
`eloquanotification.email.subject`	Subject of the notification message.
`eloquanotification.email.body`	Body of the notification message.
`eloquanotification.email.replyto`	E-mail address that receives replies to the notification message.
`eloquanotification.email.contenttype`	MIME content type of the notification message body.

54.3.5 Creating the Rendering Template for the EloquaForm Assets

To create the template that renders the `EloquaForm` proxy assets within WebCenter Sites pages, do the following:

1. Log in to the Admin interface on target WebCenter Sites site as the administrator.

2. Create a new template with the following parameters:

Field	Value
Name	Detail
For AssetType	EloquaForm
Usage	Element is used within an HTML page
Create Template Element	JSP
Element Logic	paste the contents of the EloquaForm/ Detail.jsp element

3. Save the new template.

54.3.6 Placing Eloqua Forms on WebCenter Sites Pages

To place an Eloqua form into a WebCenter Sites page, the page developer must have configured the page template to include one or more slots that accept EloquaForm proxy assets. Once the target pages have been configured in this manner, a WebCenter Sites contributor can place Eloqua forms into them as follows:

1. In the Contributor interface, create a new or open an existing page for editing.

2. In the tree on the left, select the **Content Tree** tab and expand the **Eloqua** node. (You may also perform a search for assets of type EloquaForm.)

3. Locate the desired EloquaForm asset and drag it into the designated slot on the page.

4. Make any further desired changes to the page, then save the page to commit them.

Part XV

Controlling the Site Capture Process

You can control a crawler's site capture process by implementing methods and interfaces of the `BaseConfigurator` class.

Topic:

- [Coding the Crawler Configuration File](#)

55

Coding the Crawler Configuration File

The `BaseConfigurator` class, its methods, and interfaces control a crawler's site capture process. A sample code is available in the Site Capture installation for the FirstSiteII crawler.

Topics:

55.1 About Controlling a Crawler

To control a crawler, you need to code its `CrawlerConfigurator.groovy` file with, at minimum, the starting URI and link extraction logic. You supply this information through the `getStartUri()` and `createLinkExtractor()` methods. You can also add additional code to specify, for example, the number of links to be crawled, the crawl depth, and the invocation of a post-crawl event such as copying statically downloaded files to a web server's doc base.

The methods and interfaces you use are provided in the `BaseConfigurator` class. The default implementations can be overridden to customize and control a crawl process in a way that agrees with the structure of the target site and the data you have to collect.

The `BaseConfigurator` methods and a simple `CrawlerConfigurator.groovy` file described in the topics that follow demonstrate the usage of the required methods. Crawler customization methods are then discussed and followed by information

about Site Capture's Java interfaces, including their default and custom implementations.

55.2 BaseConfigurator Methods

The `CrawlerConfigurator.groovy` file contains the code of the `CrawlerConfigurator` class. This class must extend `BaseConfigurator`, which is an abstract class that provides default implementations for the crawler.

This table lists the methods and interfaces of the `BaseConfigurator` class:

Table 55-1   Methods in the `BaseConfigurator` Class

Method Type	Method	Notes
Required	getStartUri	N/A
Required	createLinkExtractor	Factory method in the LinkExtractor interface.[1],[2]
Crawler Customization	getMaxLinks	N/A
Crawler Customization	getMaxCrawlDepth	N/A
Crawler Customization	getConnectionTimeout	N/A
Crawler Customization	getSocketTimeout	N/A
Crawler Customization	getPostExecutionCommand	N/A
Crawler Customization	getNumWorkers	N/A
Crawler Customization	getUserAgent	N/A
Crawler Customization	createResourceRewriter	Factory method in the ResourceRewriter interface.a,b

[1] The listed interfaces have default implementations, described in this chapter.

[2] Site Capture provides a sample link extractor and resource rewriter, both used by the FirstSiteII sample crawler. See Writing and Deploying a Custom Link Extractor and Writing a Custom ResourceRewriter.

This topic includes the following:

- getStartUri

- createLinkExtractor

55.2.1 getStartUri

This method injects the crawler's start URI. Configure one or more start URIs for the crawl if the URIs belong to the same site. Multiple starting points enable the crawls to start in parallel.

To provide the start URI for the `www.example.com` site:

```
/**
 * The method is used to configure the site url which needs to be crawled.
 */
public String[] getStartUri()
{
return ["http://www.example.com/home"]; //Groovy uses brackets for an array.
}
```

To provide multiple start URIs for the site, enter a comma-separated array:

```
/**
 * The method is used to configure the site url which needs to be crawled.
 */
public String[] getStartUri()
{
return ["http://www.example.com/product","http://www.example.com/support"]; //Groovy
uses brackets for an array.
}
```

55.2.2 createLinkExtractor

This method configures the logic for extracting links from the crawled pages. The extracted links are then traversed. It shows a basic sample `CrawlerConfigurator.groovy` file.

Two abstract methods in `BaseConfigurator` must be overridden in `CrawlerConfigurator`. They are `getStartUri()` and `createLinkExtractor()`. The `createLinkExtractor` method is a factory method in the `LinkExtractor` interface:

- Implement the `LinkExtractor` interface to create your own link extraction algorithm, for example, using an HTML parser to parse the pages and extract links for the crawler to consume.

- To extract links, use the default implementation, `PatternLinkExtractor`, which uses regular expressions. For example, `PatternLinkExtractor` can be used to extract links of the format `/home/products` from expressions such as `<a href="/home/product">Products</a>`.

  To use a regular expression for extracting links from `<a href="/home/product">Products</a>` on the `www.example.com` site:

  ```
  /**
   * The method is used to define the link extraction
   * algorithm from the crawled pages.
   * PatternLinkExtractor is a regex based extractor
   * which parses the links on the web page
   * based on the pattern configured inside the constructor.
   */
  public LinkExtractor createLinkExtractor()
  {
  return new PatternLinkExtractor("['\"\\(](/[^\\s<'\"\\)]*)",1);
  }
  ```

- For more information about regular expressions and `PatternLinkExtractor`, see Using the Default Implementation of LinkExtractor.

- For more information about implementing the `LinkExtractor` interface, see Writing and Deploying a Custom Link Extractor.

Basic Configuration File

This example of a simple `CrawlerConfigurator.groovy` file, the required methods, `getStartUri()` and `createLinkExtractor()`, are overridden.

In this example, we override an additional method `getMaxLinks()`. In the example, it is set to return `150` so that the test run can be completed quickly.

The file named `CrawlerConfigurator.groovy` is used to inject dependency. Hence, its name must not be changed.

```groovy
package com.fatwire.crawler.sample

import java.text.DateFormat;
import java.text.SimpleDateFormat;

import java.util.regex.Pattern;

import javax.mail.internet.AddressException;
import javax.mail.internet.InternetAddress;

import com.fatwire.crawler.*;
import com.fatwire.crawler.remote.*;
import com.fatwire.crawler.remote.di.*;
import com.fatwire.crawler.impl.*;
import com.fatwire.crawler.util.FileBuilder;

import org.apache.commons.lang.SystemUtils;
import org.apache.http.HttpHost;
import org.apache.http.auth.*;
import org.apache.http.client.*;
import org.apache.http.impl.client.*;
/**
 * Configurator for the crawler.
 * This is used to inject the dependency inside the crawler
 * to control the crawling process
 */

public class CrawlerConfigurator extends BaseConfigurator {

public CrawlerConfigurator(GlobalConfigurator delegate){
super(delegate);
}

/**
 * The method is used to configure the site url which needs to be crawled.
 */
public String[] getStartUri() {
return ["http://www.fatwire.com/home"]; //Groovy uses brackets for an array.
}

/**
 * The method is used to define the link extraction algorithm
 * from the crawled pages.
 * PatternLinkExtractor is a regex based extractor which parses
 * the links on the web page
 * based on the pattern configured inside the constructor.
 */
public LinkExtractor createLinkExtractor() {
return new PatternLinkExtractor("['\"\\(](/[^\\s<'\"\\)]*)",1);
}

/**
 * The method is used to control the maximum number of links
```

```
 * to be crawled as part of this crawl session.
 */
public int getMaxLinks()
{
150;
}
```

55.3 Crawler Customization Methods

In addition to the required methods, the `BaseConfigurator` class has methods with default implementations. You may want to override these methods to customize the crawl process in a way that agrees with the structure of the target site and the data you have to collect.

See these topics:

- getMaxLinks

- getMaxCrawlDepth

- getConnectionTimeout

55.3.1 getMaxLinks

This method controls the number of links to be crawled. The number of links should be a positive integer. Otherwise, the crawl scans all the links in the same domain that are reachable from the start URI(s).

To specify crawling 500 links:

```
/**
 * default: -1; crawler will crawl over all the links reachable from the start URI
 * @return the maximum number of links to download.
 */
public int getMaxLinks()
{
return 500;
}
```

55.3.2 getMaxCrawlDepth

This method controls the maximum depth to which a site is crawled. Links beyond the specified depth are ignored. The depth of the starting page is 0.

```
/**
 * default: -1. Indicates infinite depth for a site.
 * @return the maximum depth to which we need to crawl the links.
 */
public int getMaxCrawlDepth()
{
return 4;
}
```

55.3.3 getConnectionTimeout

This method determines how long the crawler will wait to establish a connection to its target site. If a connection is not established within the specified time, the crawler will ignore the link and continue to the next link.

To set a connection timeout of 50,000 milliseconds:

```
/**
 * default: 30000 ms
 * @return Connection timeout in milliseconds.
 */
public int getConnectionTimeout()
{
return 50000; // in milliseconds
}
```

55.4 getSocketTimeout

This method controls the socket timeout of the request that is made by the crawler for the link to be crawled.

To provide a socket timeout of 30,000 milliseconds:

```
/**
 * default: 20000 ms
 * @return Socket timeout in milliseconds.
 */
public int getSocketTimeout()
{
return 30000; // in milliseconds
}
```

55.5 getPostExecutionCommand

This method injects custom post-crawl logic, and it's invoked when the crawler finishes its crawl session. It must return the absolute path of the script or command and parameters (if any).

For example, the getPostExecutionCommand() can be used to automate deployment to a web server's doc base by invoking a batch or shell script to copy statically captured files after the crawl session ends.

Note:

- The script or command should be present in the same location on all servers hosting Site Capture.

- Avoid downloading large archive files (exceeding 250MB) from the Site Capture interface. Use getPostExecutionCommand to copy the files from the Site Capture file system to your preferred location. Archive size can be obtained from the crawler report, on the Job Details form.

To run a batch script named copy.bat on the Site Capture server:

```
/**
 * default: null.
 * @return the command string for post execution.
 * Null if there is no such command.
 */
public String getPostExecutionCommand()
{
// The file is supposed to be at the path C:\\commands folder
// on the computer where the site capture server is running
return "C:\\commands\\copy.bat";
}
```

55.6 getNumWorkers

This method controls the number of worker threads used for the crawl process. The ideal number of parallel threads to be spawned for the crawl session depends on the architecture of the computer on which Site Capture is hosted.

To start 10 worker threads for a crawl process:

```
/**
 * default: 4.
 * @return the number of workers to start.
 * Workers will concurrently downloads resources.
 */
public int getNumWorkers()
{
// Start 10 worker threads which is involved in the crawl process.
return 10;
}
```

55.7 getUserAgent

This method configures the user agent that the crawler uses when it traverses the site. You should use this method to render the site in a different way than usual. For example, to render the site on a mobile device.

To configure the FireFox 3.6.17 user agent:

```
/**
 * default: publish-crawler/1.1 (http://www.fatwire.com)
 * @return the user agent identifier
 */
public String getUserAgent()
{
return "Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US;rv:1.9.2.17) Gecko/20110420
Firefox/3.6.17 ";
}
```

55.8 createResourceRewriter

This method rewrites URLs inside the HTML pages that are crawled. For example, you may want to rewrite the URLs to enable static delivery of a dynamic WebCenter Sites website.

The `createResourceRewriter` method is a factory method in the `ResourceRewriter` interface:

- Implement the `ResourceRewriter` interface to convert dynamic URLs to static URLs, absolute URLs to relative URLs, and so on.

- You can also use the following default implementations:

    - `NullResourceRewriter`: Does not rewrite any of the URLs.

    - `PatternResourceRewriter`: Searches for a regular pattern and rewrites as specified.

To use `PatternResourceRewriter` to rewrite URLs such as `http://www.site.com/home.html` to `/home.html`:

```
/**
 * Factory method for a ResourceRewriter.
```

```
 * default: new NullResourceRewriter();
 * @return the rewritten resource modifies the html before it is saved to disk.
 */
public ResourceRewriter createResourceRewriter()
{
new PatternResourceRewriter("http://www.site.com/([^\\s'\"]*)", '/$1');
}
```

- For more information about the default implementations, see Using the Default Implementations of ResourceRewriter.

- For more information about implementing the ResourceRewriter interface, see Writing a Custom ResourceRewriter.

55.9 createMailer

This method provides the implementation for sending email after the crawl. The createMailer method is a factory method in the Mailer interface.

- Site Capture comes with an SMTP over TLS implementation, which emails the crawler report when a static or archive capture session ends (the crawler report is the report.txt file, described in *Oracle Fusion Middleware Administering Oracle WebCenter Sites*).

- If you are using a mail server other than SMTP-TLS (such as SMTP without authentication, or POP3), you must provide your own implementation.

To send no email:

```
/**
 * Factory method for a Mailer.
 * <p/>
 * default: new NullMailer().
 * @return mailer holding configuration to send an email at the end of the crawl.
 * Should not be null.
 */
public Mailer createMailer()
{
return new NullMailer();
}
```

- For more information about the default implementation, see Using the Default Implementation of Mailer.

- For more information about implementing the ResourceRewriter interface, see Writing a Custom Mailer.

55.10 getProxyHost

The getProxyHost method must be overridden if the site being crawled is behind a proxy server. You can configure the proxy server in this method.

> **Note:**
>
> If you use getProxyHost, also use getProxyCredentials, described on getProxyCredentials.

To configure a proxy server:

```
/**
 * default: null.
 * @return the host for the proxy,
 * null when there is no proxy needed
 */
public HttpHost getProxyHost()
{
//using the HTTPClient library return a HTTPHost
return new HttpHost("www.myproxyserver.com", 883);
}
```

55.11 getProxyCredentials

This method injects credentials for the proxy server which is configured in the getProxyHost method.

See getProxyHost.

To authenticate a proxy server user named sampleuser:

```
/**
 * default: null.
 * example: new UsernamePasswordCredentials(username, password);
 * @return user credentials for the proxy.
 */
public Credentials getProxyCredentials()
{
return new UsernamePasswordCredentials("sampleuser", "samplepassword");
//using the HTTPClient library return credentials
}
```

55.12 Interfaces

Site Capture provides these interfaces with default implementations: LinkExtractor, ResourceRewriter, and Mailer. Read further to know more about these interfaces.

- LinkExtractor

- ResourceRewriter

- Mailer

55.12.1 LinkExtractor

A link extractor specifies which links are traversed by Site Capture in a crawl session. The implementation is injected through the CrawlerConfigurator.groovy file. The implementation is called by the Site Capture framework during the crawl session to extract links from the markup that is downloaded as part of the crawl session.

Site Capture comes with one implementation of LinkExtractor. You can also write and deploy your own custom link extraction logic. For more information, see the following sections:

- LinkExtractor Interface

- Using the Default Implementation of LinkExtractor

- Writing and Deploying a Custom Link Extractor

55.12.1.1 LinkExtractor Interface

This interface has only one method (`extract`) that needs to be implemented to provide the algorithm for extracting links from downloaded markup.

```
package com.fatwire.crawler;
import java.util.List;
import com.fatwire.crawler.url.ResourceURL;

/**
 * Extracts the links out of a WebResource.
 */

public interface LinkExtractor
{
/**
 * Parses the WebResource and finds a list of links (if possible).
 * @param resource the WebResource to inspect.
 * @return a list of links found inside the WebResource.
 */
List<ResourceURL> extract(final WebResource resource);
}
```

55.12.1.2 Using the Default Implementation of LinkExtractor

`PatternLinkExtractor` is the default implementation for the `LinkExtractor` interface. `PatternLinkExtractor` extracts links on the basis of a regular expression. It takes a regular expression as input and returns only links matching that regular expression.

Common usage scenarios include using `PatternLinkExtractor` for sites with dynamic URLs and using `PatternLinkExtractor` for sites with static URLs.

- Using `PatternLinkExtractor` for sites with dynamic URLs:

  For example, on `www.example.com`, the links have a pattern of `/home`, `/support`, and `/cs/Satellite/`. To extract and traverse such kinds of links, use `PatternLinkExtractor` in the following way:

  ```
  /**
   * The method is used to define the link extraction algorithm
   * from the crawled pages.
   * PatternLinkExtractor is a regex based extractor which parses
   * the links on the web page
   * based on the pattern configured inside the constructor.
   */
  public LinkExtractor createLinkExtractor()
  {
  return new PatternLinkExtractor("['\"\\(](/[^\\s<'\"\\)]*)",1);
  }
  ```

  The pattern **[' \ " \ \ ( ] (/ [^ \ \ s < ' \ " \ \ )] *)** is used to extract links:

  - that start with any one of the following characters:

    * Single quote (')

    * Double quotes (")

    * Left parenthesis ((

      –   continue with a slash (/),

      –   and end with any one of the following characters:

          *   Spaces (\s)

          *   Less-than symbol (<)

          *   Single quote (')

          *   Double quote (")

          *   Right parenthesis)

Let's consider the URL inside the following markup:

```
<a href='/home'>Click Me</a>
```

We are interested only in extracting the `/home` link. This link matches the regular expression pattern because it starts with a single quote (') and ends with a single quote ('). The grouping of 1 will return the result as `/home`.

- Using `PatternLinkExtractor` for sites with static URLs:

  For example, the markup for `www.example.com` has links such as:

  ```
  <a href="http://www.example.com/home/index.html">Click
  Me</a>
  ```

  To extract and traverse such types of links, we can use `PatternLinkExtractor` in the following way:

  ```
  /**
   * The method is used to define the link extraction algorithm
   * from the crawled pages.
   * PatternLinkExtractor is a regex based extractor which parses
   * the links on the web page
   * based on the pattern configured inside the constructor.
   */
  public LinkExtractor createLinkExtractor()
  {
  return new PatternLinkExtractor(Pattern.compile("http://www.example.com/[^\
  \s<'\"]*"));
  }
  ```

  The above example instructs the crawler to extract links that start with `http://www.example.com` and end with any one of the following characters: spaces (\s), less-than symbol (<), single quote ('), or double quotes (").

  Note:

  For more details on groups and patterns, refer to the Java documentation for the `Pattern` and `Matcher` classes.

55.12.1.3 Writing and Deploying a Custom Link Extractor

Site Capture provides a sample link extractor (and resource rewriter) used by the FirstSiteII sample crawler to download WebCenter Sites FirstSiteII dynamic website as a static site. For more information, see the source code for the `FSIILinkExtractor` class in the following folder:

```
<SC_INSTALL_DIR>/fw-site-capture/crawler/_sample/FirstSiteII/src
```

To write a custom link extractor:

1. Create a project in your Java IDE.

2. Copy the file `fw-crawler-core.jar` from the `<SC_INSTALL_DIR>/fw-site-capture/webapps/ROOT/WEB-INF/lib` folder to your project's build path.

3. Implement the `LinkExtractor` interface to provide the implementation for the `extract()` method. (The `LinkExtractor` interface is shown on LinkExtractor Interface.)

   Below is pseudo-code showing a custom implementation:

```
package com.custom.crawler;
import java.util.List;
import com.fatwire.crawler.url.ResourceURL;
import com.fatwire.crawler.LinkExtractor;
/**
 * Extracts the links out of a WebResource.
 */
public class CustomLinkExtractor implements LinkExtractor
{
/**
 * A sample constructor for CustomLinkExtractor
 */
public CustomLinkExtractor(String .......)
  {
  // Initialize if there are private members.
  // User's custom logic
  }

/**
 * Parses the WebResource and finds a list of links (if possible).
 * @param resource the WebResource to inspect.
 * @return a list of links found inside the WebResource.
 */
List<ResourceURL> extract(final WebResource resource)
  {
  // Your custom code for extraction Algorithm.
  }
}
```

4. Create a `jar` file for your custom implementation and copy it to the following folder:

```
<SC_INSTALL_DIR>/fw-site-capture/webapps/ROOT/WEB-INF/lib
```

5. Restart the Site Capture application server.

6. Inject the dependency by coding the `CrawlerConfigurator.groovy` file to include the custom link extractor class (`CustomLinkExtractor`, in this example):

```
/**
 * User's custom link extractor mechanism to extract the links from the
 * web resource downloaded as part of the crawl session.
 * The code below is only a pseudo code for an example.
 * User is free to implement their own custom constructor
 * as shown in the next example.
 */
public LinkExtractor createLinkExtractor()
```

```
{
return new CustomLinkExtractor("Custom Logic For Your Constructor");
}
```

55.12.2 ResourceRewriter

A resource rewriter rewrites URLs inside the markup that is downloaded during the crawl session. The implementation must be injected through the `CrawlerConfigurator.groovy` file.

Some use cases that require a resource rewriter are:

- Crawling a dynamic site and creating a static copy.

- Converting absolute URLs to relative URLs. For example, if the markup has URLs such as `http://www.example.com/abc.html`, then the crawler should remove `http://www.example.com` from the URL, thus allowing resources to be served from the host on which the downloaded files are stored.

Site Capture comes with the two implementations of `ResourceRewriter`. You can also create custom implementations. For more information, see the following sections:

- ResourceRewriter Interface

- Using the Default Implementations of ResourceRewriter

- Writing a Custom ResourceRewriter

55.12.2.1 ResourceRewriter Interface

The `rewrite` method rewrites URLs inside the markup that is downloaded during the crawl session.

```
package com.fatwire.crawler;
import java.io.IOException;


/**
 * Service for rewriting a resource. The crawler will use the implementation for
 * rewrite method to rewrite the resources that are downloaded as part of crawl
 * session.
 */
public interface ResourceRewriter
{
/**
 * @param resource
 * @return the bytes after the rewrite.
 * @throws IOException
 */
byte[] rewrite(WebResource resource) throws IOException;

}
```

55.12.2.2 Using the Default Implementations of ResourceRewriter

Site Capture comes with the following implementations of `ResourceRewriter`:

- `NullResourceRewriter`, configured by default to skip the rewriting of links. If `ResourceRewriter` is not configured in the `CrawlerConfigurator.groovy` file, then `NullResourceRewriter` is injected by default.

- `PatternResourceRewriter`, used to rewrite URLs based on the regular expression. `PatternResourceRewriter` takes as input a regular expression to match the links inside the markup and replaces those links with the string that is provided inside the constructor.

  To rewrite an absolute URL as a relative URL:

  From:

  ```
  <a href="http://www.example.com/about/index.html">Click
  Me</a>
  ```

  To:

  ```
  <a href="/about/index.html">Click Me</a>
  ```

  ```
  /**
   * Factory method for a ResourceRewriter.
   * default: new NullResourceRewriter();
   * @return the rewritten resource modifies the html before it is saved to disk.
   */
  public ResourceRewriter createResourceRewriter()
  {
  new PatternResourceRewriter("http://www.example.com/([^\\s'\"]*)", '/$1');
  }
  ```

  `PatternResourceRewriter` has only one constructor that takes a regular expression and a string replacement:

  ```
  PatternResourceRewriter(final String regex, final String
  replacement)
  ```

55.12.2.3 Writing a Custom ResourceRewriter

Site Capture provides a sample resource rewriter (and link extractor) used by the FirstSiteII sample crawler to download WebCenter Sites' FirstSiteII dynamic website as a static site. For more information, see the source code for the `FSIILinkExtractor` class in the following folder:

```
<SC_INSTALL_DIR>/fw-site-capture/crawler/_sample/FirstSiteII/src
```

To write a custom resource rewriter:

1. Create a project in your IDE.

2. Copy the files `fw-crawler-core.jar` from `<SC_INSTALL_DIR>/fw-site-capture/webapps/ROOT/WEB-INF/lib` folder to your project's build path.

3. Implement the `ResourceRewriter` interface to provide the implementation for the `rewrite` method.

   Below is a pseudo-code showing a custom implementation:

   ```
   package com.custom.crawler;
   import com.fatwire.crawler.WebResource;
   import com.fatwire.crawler.ResourceRewriter;

   /**
    * Rewrite the links inside the markup downloaded as part of
    * crawl session.
    */
   public class CustomResourceRewriter implements ResourceRewriter
   {
   /**
   ```

```
 * A sample constructor for CustomResourceRewriter
 */
public CustomResourceRewriter(String .......)
  {
  // Initialize if there are private members.
  // User's custom logic
  }

/**
 * @param resource
 * @return the bytes after the rewrite.
 * @throws IOException
 */
byte[] rewrite(WebResource resource) throws IOException
  {
  // Your custom code for re-writing Algorithm.
  }
}
```

4. Create a `jar` file for your custom implementation and copy it to the following folder: `<SC_INSTALL_DIR>/fw-site-capture/webapps/ROOT/WEB-INF/lib`

5. Restart your Site Capture application server.

6. Inject the dependency by coding the `CrawlerConfigurator.groovy` file to include the custom `resource rewriter` class (`CustomResourceRewriter`, in this example):

```
/*
 * User's custom resource rewriting mechanism to rewrite the links from the
 * web resource downloaded as part of the crawl session.
 *
 * The code below is only a pseudo code for an example.
 * User is free to implement their own custom constructor
 * as shown in the next example.
 */
public ResourceRewriter createResourceRewriter()
  {
  new CustomResourceRewriter("User's custom logic to initialize the things");
  }
```

55.12.3 Mailer

A mailer sends email after the crawl ends. The implementation must be injected through the `CrawlerConfigurator.groovy` file.

Site Capture provides an `SMTPTlsMailer` implementation, which can be used to send the crawler report from the SMTP-TLS mail server. You also can implement the `Mailer` interface to provide custom logic for sending emails from a server other than SMTP-TLS (such as SMTP without authentication, or POP3). Your custom logic also can specify the email to be an object other than the crawler report. If `Mailer` is not configured in the `CrawlerConfigurator.groovy` file, then `NullMailer` is injected by default.

This section includes the following topics:

• Mailer Interface

• Using the Default Implementation of Mailer

- [Writing a Custom Mailer](#)

55.12.3.1 Mailer Interface

The `sendMail` method is automatically called if the `Mailer` is configured in the `CrawlerConfigurator.groovy` file.

```
package com.fatwire.crawler;
import java.io.IOException;
import javax.mail.MessagingException;

/**
 * Service to send an email.
 */
public interface Mailer
{
/**
 * Sends the mail.
 *
 * @param subject
 * @param report
 * @throws MessagingException
 * @throws IOException
 */
void sendMail(String subject, String report)
throws MessagingException, IOException;
}
```

55.12.3.2 Using the Default Implementation of Mailer

Site Capture provides an SMTP-TLS server-based email implementation that sends out the crawler report when a static or archive crawl session ends. (The crawler report is a (`report.txt` file, described in About Accessing Log Files in *Oracle Fusion Middleware Administering Oracle WebCenter Sites*).

- Use the default mailer by injecting it through the `CrawlerConfigurator.groovy` file, as shown below:

```
/**
 * Factory method for a Mailer.
 * <p/>
 * default: new NullMailer().
 * @return mailer holding configuration to send an email
 * at the end of the crawl.
 * Should not be null.
 */
public Mailer createMailer()
{
 try
  {
  // Creating a SmtpTlsMailer Object
  SmtpTlsMailer mailer = new SmtpTlsMailer();

  InternetAddress from;
  // Creating an internet address from whom the mail
  // should be sent from = new InternetAddress("example@example.com");

  // Setting the mail address inside the mailer object mailer.setFrom(from);

  // Setting the email address of the recipient inside
  // mailer.mailer.setTo(InternetAddress.parse("example@example.com"));
```

```
      // Setting the email server host for to be used for email.
      // The email server should be SMTP-TLS enabled.
      mailer.setHost("smtp.gmail.com", 587);

      // Setting the credentials of the mail account
      // mailer.setCredentials("example@example.com", "examplepassword");

      return mailer;
      }
   catch (AddressException e)
      {
      log.error(e.getMessage());
      }
   }
```

55.12.3.3 Writing a Custom Mailer

This section provides the steps to write a custom mailer.

To write a custom mailer:

1. Create a project in your IDE.

2. Copy the files `fw-crawler-core.jar` from the `<SC_INSTALL_DIR>/fw-site-capture/webapps/ROOT/WEB-INF/lib` folder in your project's build path.

3. Implement the `Mailer` interface with the `sendMail` method.

   Below is a pseudo-code showing a custom implementation:

```
package com.custom.crawler;
import java.io.IOException;
import javax.mail.MessagingException;
import com.fatwire.crawler.Mailer;

/**
 * Implements an interface to implement the logic for sending emails
 * when the crawl session has been completed.
 */
public class CustomMailer implements Mailer
{
/**
 * A sample constructor for CustomMailer
 */
public CustomMailer()
   {
   // Initialize if there are private members.
   // User's custom logic
   }

/**
 * Sends the mail.
 *
 * @param subject
 * @param report
 * @throws MessagingException
 * @throws IOException
 */
void sendMail(String subject, String report)
throws MessagingException, IOException
```

```
     {
     // User's custom logic to send the emails.
     }
   }
```

4. Create a `jar` file for your custom implementation and copy it to the following folder: `<SC_INSTALL_DIR>/fw-site-capture/webapps/ROOT/WEB-INF/lib`

5. Restart your Site Capture application server.

6. Inject the dependency by coding the `CrawlerConfigurator.groovy` file to include the custom mailer class (`CustomMailer`, in this example):

```
/**
 * Factory method for a Mailer.
 * <p/>
 * default: new NullMailer().
 * @return mailer holding configuration to send an email
 * at the end of the crawl.
 * Should not be null.
 */
public Mailer createMailer()
{
CustomMailer mailer = new CustomMailer();
// Do some of the initilization stuffs
return mailer;
}
package com.custom.crawler;
import java.io.IOException;
import javax.mail.MessagingException;
import com.fatwire.crawler.Mailer;

/**
 * Implements an interface to implement the logic for sending emails
 * when the crawl session has been completed.
 */
public class CustomMailer implements Mailer
{
  /**
   * A sample constructor for CustomMailer
   */
  public CustomMailer()
    {
    // Initialize if there are private members.
    // User's custom logic
    }
  /**
   * Sends the mail.
   *
   * @param subject
   * @param report
   * @throws MessagingException
   * @throws IOException
   */
  void sendMail(String subject, String report)
  throws MessagingException, IOException
    {
    // User's custom logic to send the emails.
    }
}
```

This implementation emails the crawler report (the `report.txt` file), given that the `String report` argument in the `sendMail` method names the crawler report, by default. You can customize the logic for emailing objects other than the crawler report.

55.13 Summary of Methods and Interfaces

For controlling a crawler's site capture process, the default implementations of methods and interfaces in the Site Capture `BaseConfigurator` class are described here.

See these topics:

- **Methods**

- Interfaces

55.13.1 Methods

The following interfaces are used in the Site Capture `BaseConfigurator` class:

- getStartUri

- createLinkExtractor

- getMaxLinks

- getMaxCrawlDepth

- getConnectionTimeout

- getSocketTimeout

- getPostExecutionCommand

- getNumWorkers

- getUserAgent

- createResourceRewriter

- createMailer

- getProxyHost

- getProxyCredentials

The factory methods are in the following interfaces:

- createLinkExtractor is in the LinkExtractor interface.

- createResourceRewriter is in the ResourceRewriter interface.

- createMailer is in the Mailer interface.

55.13.2 Interfaces

The following interfaces are used in the Site Capture `BaseConfigurator` class:

- LinkExtractor

  Its default implementation is `PatternLinkExtractor`, which extracts links on the basis of a regular expression.

Site Capture also provides a sample link extractor (and a sample resource rewriter), used by the FirstSiteII sample crawler to download WebCenter Sites' FirstSiteII dynamic website as a static site. Source code is available in the following folder: `<SC_INSTALL_DIR>/fw-site-capture/crawler/_sample/FirstSiteII/src`

You can write and deploy your own custom link extraction logic.

- ResourceRewriter

Its default implementations are `NullResourceRewriter`, which skips the rewriting of links, and `PatternResourceRewriter`, which rewrites URLs based on the regular expression.

Site Capture provides a sample resource rewriter (and a sample link extractor), used by the FirstSiteII sample crawler to download WebCenter Sites' FirstSiteII dynamic website as a static site. Source code is available in the following folder:`<SC_INSTALL_DIR>/fw-site-capture/crawler/_sample/FirstSiteII/src`

You can write and deploy your own logic for rewriting URLs.

- Mailer

Its default implementation is `SMTPTlsMailer`, which sends the crawler report from the SMTP-TLS mail server. You can customize the logic for emailing other types of objects from other types of servers.

Part XVI

Integrating with Third-Party Content Sources

You can use Proxy assets to extend the Content Integration Platform to publish from systems of your own choice to Oracle WebCenter Sites.

- Integrating Third-Party Content Sources Using Proxy Assets

56

Integrating Third-Party Content Sources Using Proxy Assets

Content and marketing teams often need to publish content that resides outside websites developed with WebCenter Sites. You can enable them to integrate external web content using the proxy asset type framework.

Topics:

- Proxy Asset Architecture and the Contributor Interface

- Installing Sample Proxy Assets

- Integrating External Content in the Contributor Interface

- Setting Up YouTube Proxy Assets

- User Interface Customizations

- Information About Embedding Proxy Assets in Web Pages

56.1 Proxy Asset Architecture and the Contributor Interface

In the proxy asset architecture, you make a third-party content repository accessible from the editorial and delivery instances and customize the contributor interface to access this repository.

Proxy Asset Architecture

A proxy asset is an asset representing content stored and managed in a remote location. This figure shows the proxy asset architecture:

Figure 56-1   Proxy Asset Architecture
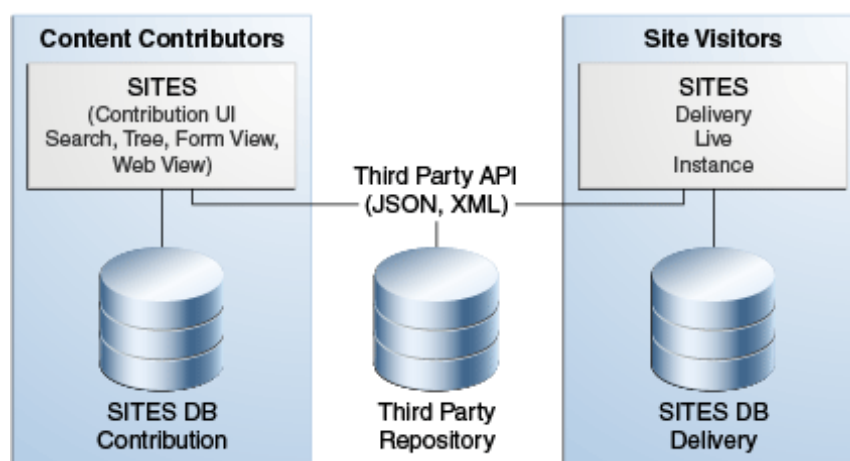
In this architecture:

- A third-party content repository is assumed, accessible through a public read API from both the editorial and delivery instances.

- The Contributor interface is customized to access the third-party repository to make external content visible in the UI.

- A proxy asset is created on the fly for every external content rendered in the UI. A proxy asset does not store any metadata, which is assumed to be accessible through a public read API provided by the third-party service.

- On the live instance, templates written for proxy assets are accessing the same repository and API to render external content on the live site.

> **Note:**
>
> Only those proxy assets that are used are permanently stored in the WebCenter Sites database. For example, proxy assets created while rendering search results are later purged by a dedicated cleaning event.

Contributor Interface

Once a new proxy asset type is registered in the WebCenter Sites database, developers have to provide the appropriate UI customizations before contributors can start interacting with external content.

The nature of the UI customizations being implemented depend on the contributor's specific requirements. For example, it is expected that the Contributor interface search functionality is hooked to the external repository's own search service in most cases.

After this is done, and external content is surfaced in the Contributor interface in the form of proxy assets, they behave mostly like standard assets. That is, contributors are able to:

- Search the external content repository, using the same asset search tab, showing results in list or thumbnail view, docked or undocked.

- Use drag and drop from search or tree.

- Associate external content to other assets, whether in form view or web view.

- Preview external content, using WebCenter Sites templates.

- Bookmark, tag, set in workflow, approve, and publish.

> **Note:**
>
> The following restrictions apply:
>
> - The external content lifecycle is assumed to be entirely managed in a third-party UI; that is, WebCenter Sites only needs read access to the external repository. Consequently, UI actions such as editing, versioning, and so forth, are disabled by default for all proxy asset types.
>
> - Proxy assets cannot have associated assets or subtypes.
>
> - The external content repository must be accessible from both the contribution and delivery WebCenter Sites instances.

56.2 Installing Sample Proxy Assets

To install proxy assets in Oracle WebCenter Sites, you begin with setting up a proxy asset directory and creating a proxy asset.

- Set up a Proxy Asset Directory

- Create a Proxy Asset

- Add the Search Functionality for the Proxy Asset

- Add the Thumbnail Grid Functionality for the Proxy Asset

- Add the Tree Functionality for the Proxy Asset

> **Note:** See Integrating External Content in the Contributor Interface.

56.2.1 Set up a Proxy Asset Directory

To set up a proxy asset directory:

1. In your Oracle WebCenter Sites installation, navigate to `sites-home\bootstrap \samples\miscellaneous\SampleProxy\proxy_sample`. In the subsequent steps this directory is referred to as `${WCS_PROXY}`.

2. Copy the `proxy_samples` directory under the web application root on the server. For example, `sites-webapp-server\webapps\sites\proxy_samples`.

    This directory contains the sample JSON and images to simulate a third-party API.

3. To test the `proxy_samples` directory, start the server and navigate to `http:// localhost:8080/sites/proxy_samples/search/ski.json`. Then click one of the files in your browser to see the JSON text.

56.2.2 Create a Proxy Asset

To create a proxy asset:

1. Log into WebCenter Sites and launch the Admin interface.

2. Expand **Admin**, then **Proxy Asset Maker**, and then double click **Add New**.

3. In the form, enter values in these fields: **Name** (e.g: `ProxyTest`), **Description**, and **Plural Form** (e.g.: `ProxyTests`) for the proxy asset.

4. Click **Save**.

5. Enable the `ProxyTest` asset in your site. To do this, create a **Search** start menu for `ProxyTest`.

    You cannot yet access the ProxyTest Search start menu in the Contributor interface.

56.2.3 Add the Search Functionality for the Proxy Asset

To add the search functionality:

1. Open Sites Explorer and log in to your WebCenter Sites instance.

2. In Sites Explorer, under `Tables/ElementCatalog/CustomElements`, create the following directory structure: `ProxyTest/UI/Data/Search`:
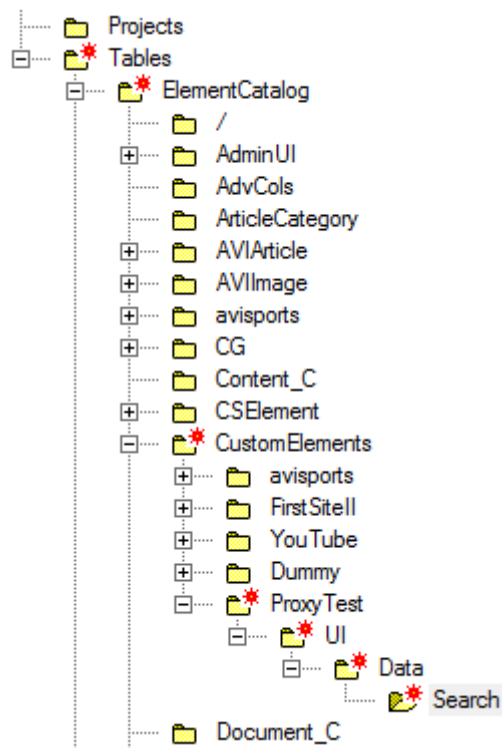
3. Under `ProxyTest/UI/Data/Search`, add two new rows.

4. Copy and paste the contents from the corresponding files in `${WCS_PROXY}\src\jsp\cs_deployed\CustomElements\ProxyTest\UI\Data\Search` and save:

   - `SearchAction`

     - elementname: `SearchAction`

     - url: `CustomElements\ProxyTest\UI\Data\Search\SearchAction.jsp`

   - `SearchJson`

     - elementname: `SearchJson`

     - url: `CustomElements\ProxyTest\UI\Data\Search\SearchJson.jsp`

5. In Sites Explorer, under **ElementCatalog**, add ProxyTest: `Tables\ElementCatalog\ProxyTest`.

6. Under `Tables\ElementCatalog\ProxyTest`, create a new entry by copying and pasting the contents from `${WCS_PROXY}\src\jsp\cs_deployed\ProxyTest\GetData.jsp`: GetData (elementname: `GetData`, url: `ProxyTest/GetData.jsp`), and then save these changes.

7. In the Contributor interface, in the **Search** field for ProxyTest, enter ski, surfing, or nothing.

You should see a list populated with proxy assets.

56.2.4 Add the Thumbnail Grid Functionality for the Proxy Asset

To add the thumbnail grid functionality:

1. In Sites Explorer, under `Tables\ElementCatalog\CustomElements` `\ProxyTest\UI`, create the following folder structure: `\Layout\CenterPane` `\Search\View`.

2. Add the following entries by copying and pasting the contents from the corresponding files in `${WCS_PROXY}\src\jsp\cs_deployed` `\CustomElements\ProxyTest\UI\Layout\CenterPane\Search\View`.

   - `ThumbnailViewConfig`:

     – elementname: `ThumbnailViewConfig`

     – url: `CustomElements\ProxyTest\UI\Layout\CenterPane\Search` `\View\ThumbnailViewConfig.jsp`

   - `DockedThumbnailViewConfig`:

     – elementname: `DockedThumbnailViewConfig`

     – url: `CustomElements\ProxyTest\UI\Layout\CenterPane\Search` `\View\DockedThumbnailViewConfig.jsp`

3. In the Contributor interface, enter a valid search term for ProxyTest assets.

   You will see the thumbnail images in the grid mode.

56.2.5 Add the Tree Functionality for the Proxy Asset

To add the tree functionality:

1. In Sites Explorer, under `Tables\ElementCatalog\ProxyTest`, create a directory called `Tree`.

2. Add the following entries by copying and pasting the contents from the corresponding files in `${WCS_PROXY}\src\jsp\cs_deployed\ProxyTest` `\Tree`.

   - `Load`

     – elementname: `Load`

     – url: `ProxyTest\Tree\Load.jsp`

   - `Root`

     – elementname: `Root`

     – url: `ProxyTest\Tree\Root.jsp`

3. Switch to the Admin interface.

4. Add a tree tab:

    a. Expand the **Admin** node.

    b. In the tree, double-click **Tree**.

        The Tree Tabs page is displayed.

    c. Click **Add New Tree Tab**.

    d. Enter the following:

- In the **Title** field, enter `ProxyTestTreeTab`.

- In the **Sites** box, choose **avisports** (or whichever site you're using).

- In the **Required Roles** box, choose **Any**.

- In the **Tab Contents** box, select **ProxyTest** and click **Add Selected Items**.

- In the selected box, click **ProxyTest**.

- In the **Section Name** field, enter `ProxyTest`.

- In the **Element Name** field, enter `ProxyTest/Tree/Root` (use only forward slashes).

- Click **Edit Section**, then click **Save**.

5. Switch to the Contributor interface.

6. On the **Content Tree**, and expand the **ProxyTestTreeTab** tree tab.

You should see the categories **ski** and **surfing** available for drag and drop, along with their respective assets.

56.3 Integrating External Content in the Contributor Interface

Contributors will be able to work with external content when you've integrated the external content repository with the repository search service and content tree.

In this topic, we'll use the ProxyTest content repository as a case study and explain the steps to integrate it with:

- **Search**: To use the repository search service instead of the standard WebCenter Sites search.

- **Content tree**: To allow contributors to browse the repository content in a custom content tree.

Topics:

- Case Study: The ProxyTest Repository

- Registering a New Proxy Asset Type

- About Implementing UI Integration Code

- Customizing Search

- Implementing a Custom Tree

56.3.1 Case Study: The ProxyTest Repository

This section describes setting up sample data and retrieving data from the ProxyTest repository.

Setting Up Sample Data

We're using a set of static JSON files deployed directly in the WebCenter Sites web application to emulate the ProxyTest content repository. The ProxyTest content is assumed to be media content (images).

Those JSON files simulate the following services:

*   Search the repository for a given term (searching on all, `ski` or `surfing` returns actual results): `http://localhost:7001/sites/proxy_samples/search/<searchterm>.json`

*   Get all content categories (Ski and Surfing): `http://localhost:7001/sites/proxy_samples/browse/categories.json`

*   Get all ProxyTest content for a given category: `http://localhost:7001/sites/proxy_samples/browse/<category>.json`

*   Get metadata for a given content ID: `http://localhost:7001/sites/proxy_samples/content/<id>.json`

For example, `/proxy_samples/search/ski.json` returns the following example content:

```
{
  "items": [
    {
      "id": "1001",
      "title": "Yellow Skier",
      "foo": "bar1",
      "lastModified": "1354735336444",
      "thumbnail": "proxy_samples/images/image7_thumbnail.png"
    },
    {
      "id": "1002",
      "title": "Female Skier",
      "foo": "bar2",
      "lastModified": "1354735336444",
      "thumbnail": "proxy_samples/images/image8_thumbnail.png"
    },
    {
      "id": "1003",
      "title": "Ski Jump",
      "foo": "bar3",
      "lastModified": "1354735336444",
      "thumbnail": "proxy_samples/images/image9_thumbnail.png"
    }
  ]
}
```

> **Note:**
>
> Sample code is stored in the folder /misc/proxy_samples/. This folder is
> located in sites-home\bootstrap\proxy_samples\miscellaneous
> \SampleProxy. Sample code specific to proxy assets is here: sites-home
> \bootstrap\samples\miscellaneous\SampleProxy\proxy_samples
> \.

Retrieving Data from the ProxyTest Repository

To avoid duplication of code, logic needed to query the external content source is
encapsulated in a dedicated element named ProxyTest/GetData. In this example,
data is returned in JSON format. Therefore, this example uses the jersey (http://
jersey.java.net/) and jettison (http://jettison.codehaus.org/) libraries,
deployed in the WebCenter Sites web application, to retrieve and deserialize incoming
JSON data.

The element receives a query, for example, /search/ski.json, in an ICS variable
named serviceURL and returns a JSONArray object stored in the ICS scope using
ics.SetObj(String, Object):

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"
%><%@ taglib prefix="ics" uri="futuretense_cs/ics.tld"
%><%@ page import="javax.ws.rs.client.*"
%><%@ page import="javax.ws.rs.core.*"
%><%@ page import="org.codehaus.jettison.json.*"
%><cs:ftcs>
<%
//
// get data from proxytest repository
//
//

Client client = ClientBuilder.newClient();
Response resp = null;
WebTarget res = null;


String host = request.getServerName();
String port = Integer.toString(request.getServerPort());
String contextPath = request.getContextPath();
String urlPath = "http://"+host+":"+port+"/"+contextPath+"/proxy_samples" +
ics.GetVar("serviceURL");


try {
      res = client.target(urlPath);
      resp = res.request(MediaType.APPLICATION_JSON).get();
}
catch(Exception e) {
      e.printStackTrace();
      throw e;
}

JSONArray list = new JSONArray();

if (resp.getStatus() == 200) {
      String jsonString = resp.readEntity(String.class);
```

```
        JSONObject json = new JSONObject(jsonString);
        list = json.getJSONArray("items");
}

ics.SetObj("items", list);

%>

</cs:ftcs>
```

56.3.2 Registering a New Proxy Asset Type

To represent ProxyTest content in the WebCenter Sites repository, define a new proxy asset type.

To create a proxy asset type:

1. On the **Admin** node, expand **Proxy Asset Manager**, and double-click **Add New**.

   The Add New Proxy Asset Type page opens.

   Figure 56-2   Add New Proxy Asset Type Page

2. In the **Name** field, enter a name for the proxy asset type. Similarly, enter a description in the **Description** field, and enter a plural form of the name in the **Plural Form** field.

   Note:

   Creating or editing proxy assets through the Contributor interface is not available. Consequently, in this step, only a Search start menu should be enabled.

   In this example, and to use with the examples continuing through the rest of the document, enter ProxyTest in the **Name** field, ProxyTest in the **Description** field, and ProxyTests in the **Plural Form** field.

3. Click **Save**.

> **Note:**
>
> Proxy Asset Maker registers the new asset type and creates a single table with the same name. A proxy asset table has only a subset of standard asset metadata, and defines only one specific column: `externalid`. This column is meant to store the identifier of the external content in the external repository.

56.3.3 About Implementing UI Integration Code

Assets can appear in many places in the Contributor interface. Some examples of these places include:

- Asset forms, for fields including asset references

- Search results

- Content tree panel

In each case, the actual integration code varies based on the requirements and available customization hooks. However, in all cases, the following principle must be followed: All external content presented in the Contributor interface must be registered as a proxy asset.

In practice, this means creating (or reusing) a proxy asset which `externalid` refers to the content identifier in the external content source. This can be done using the usual Asset API classes and methods. However, for simplicity, the proxy JSP tag library is provided.

It contains utility tags to register a given external content as a proxy asset type:

- `<proxy:register />`

It also contains utility tags to generate a JSON datastore for data grid widgets (such as search):

- `<proxy:createstore />`: Initializes a new store.

- `<proxy:addstoreitem />`: Adds an item to a given store.

- `<proxy:tojson />`: Serializes a store to JSON.

See the *Oracle Fusion Middleware Tag Reference for Oracle WebCenter Sites Reference* and the code examples in Customizing Search.

56.3.4 Customizing Search

As explained in Customizing the Search Start Menu, an override of the controller element `UI/Data/Search/Search` for the ProxyTest asset type is created using the following elements:

```
\sites-home\bootstrap\samples\miscellaneous\SampleProxy\proxy_sample\src\jsp
\cs_deployed\CustomElements\ProxyTest\UI\Data\Search\SearchAction.jsp
\sites-home\bootstrap\samples\miscellaneous\SampleProxy\proxy_sample\src\jsp
\cs_deployed\CustomElements\ProxyTest\UI\Data\Search\SearchAction.jsp\SearchJson
```

The figure below shows the **ProxyTest** option in the search drop-down. Selecting **ProxyTest** (that is, the name of the proxy asset type created) runs the custom search code, instead of the Lucene-based search.

Figure 56-3   Search Drop-Down Showing ProxyTest Asset
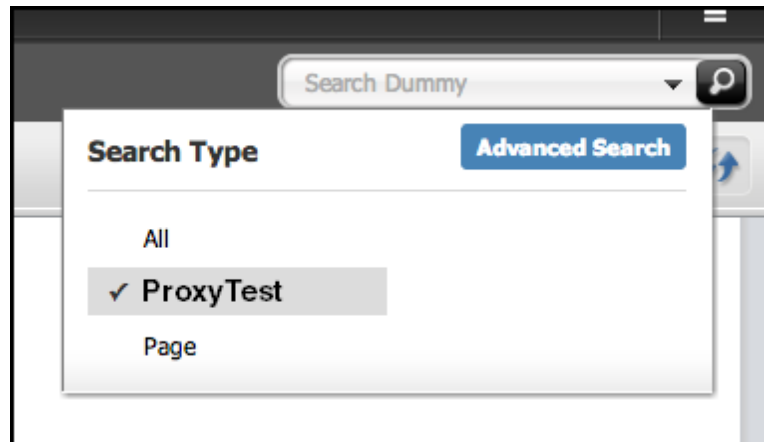
To implement a full custom search, complete the remaining topics in this section.

This topic includes the following:

- Getting Search Results Using the Provided Third-Party API

- Turning Search Results into Proxy Assets, Filter Incoming Search Results, Register External Content, and Gather Data for Search Grid Widget

- Building a Data Store for the Grid Widget

- Testing Custom Search

- Additional Customizations

56.3.4.1 Getting Search Results Using the Provided Third-Party API

1. In `CustomElements/ProxyTest/UI/Data/Search/SearchAction`, retrieve the JSON data using the element written in Case Study: The ProxyTest Repository.

   Use this code example in the retrieval:

   ```
   <%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"%>
   <%@ taglib prefix="ics" uri="futuretense_cs/ics.tld"%>
   <%@ taglib prefix="proxy" uri="futuretense_cs/proxy.tld"%>
   <%@ page import="org.codehaus.jettison.json.*"%>
   <%@ page import="COM.FutureTense.Interfaces.Utilities"%>
   <cs:ftcs>

   <%
   // in this ProxyTest example, only the empty search string, 'ski' or 'surfing'
   will
   // return results String searchTerm = ics.GetVar("searchText");
   if (!Utilities.goodString(searchTerm)) searchTerm = "all";
   %>
   <ics:setvar name="serviceURL" value='<%="/search/" + searchTerm + ".json" %>' />
   <ics:callelement element="ProxyTest/GetData" />
   <%
   JSONArray list = (JSONArray)ics.GetObj("items");

   //
   // ...to be continued in the next section
   //
   %>
   ```

```
</cs:ftcs>
```

2. At this point, `list` contains the JSON data received from the external content source.

56.3.4.2 Turning Search Results into Proxy Assets, Filter Incoming Search Results, Register External Content, and Gather Data for Search Grid Widget

This builds the code through a series of steps:

1. Build a new data store:

```
<proxy:createstore store="<storeName>" />
```

where *<storeName>* is an arbitrary string designating the data store.

2. Then, for each incoming external content asset, register the current asset as a proxy asset:

```
<proxy:register
        externalid="<external_content_identifier>"
        type="<proxy_asset_type>"
        name="<proxy_asset_name>"
        varname="<variable_name>" />
```

where:

- *<external_content_identifier>* is the identifier of the current external content item in the third-party repository. In the case of this example ProxyTest repository, this identifier is returned in the `id` of incoming JSON data.

- *<proxy_asset_type>* is the proxy asset type name. In this example, `ProxyTest`.

- *<proxy_asset_name>* is the readable string to be used as name throughout the Contributor interface. In this example, the `title` returned in the incoming JSON data.

- *<variable_name>* is the name of the WebCenter Sites variables that contain the proxy asset ID corresponding to the current external content item.

3. Then, for each incoming registered proxy asset, add the asset to the data store:

```
<proxy:addstoreitem
        store="<storeName>"
        id="<proxy_asset_id>"
        type="<proxy_asset_type>" />
```

where:

- *<storeName>* is the data store, as defined by `<proxy:createstore />`.

- *<id>* is the proxy asset identifier.

- *<type>* is the proxy asset type.

The full code for the ProxyTest proxy asset type is then:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"%>
<%@ taglib prefix="ics" uri="futuretense_cs/ics.tld"%>
<%@ taglib prefix="proxy" uri="futuretense_cs/proxy.tld"%>
<%@ page import="org.codehaus.jettison.json.*"%>
<%@ page import="COM.FutureTense.Interfaces.Utilities"%>
<cs:ftcs>

<%
// in this ProxyTest example, only the empty search string, 'ski' or 'surfing' will
// return results
String searchTerm = ics.GetVar("searchText");
if (!Utilities.goodString(searchTerm)) searchTerm = "all";
%>
<ics:setvar name="serviceURL" value='<%="/search/" + searchTerm + ".json" %>' />
<ics:callelement element="ProxyTest/GetData" />
<%
JSONArray list = (JSONArray)ics.GetObj("items");
%>

<%-- create a new data store --%>
<proxy:createstore store="assets" />
<%
// go through each incoming item
for (int i = 0; i < list.length(); i++) {
    JSONObject item = (JSONObject)list.get(i);%>

    <%-- Register the current external content item as a proxy asset --%>
    <proxy:register externalid='<%=item.getString("id") %>'
    type="ProxyTest"
    name='<%=item.getString("title") %>'
    varname="internalid" />

<%-- Add the proxy asset to the datastore --%>
<proxy:addstoreitem store="assets"
                    id='<%=ics.GetVar("internalid") %>'
                    type="ProxyTest" />
<%
}
// put store name in request scope
request.setAttribute("store", "assets");
request.setAttribute("total", Integer.valueOf(list.length()));
%>
</cs:ftcs>
```

56.3.4.3 Building a Data Store for the Grid Widget

This renders the JSON to be sent back to the UI widget.

Inside `CustomElements/ProxyTest/UI/Data/Search/SearchJson`, use the `proxy:tojson` utility tag does this, as shown in the following example:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"%>
<%@ taglib prefix="proxy" uri="futuretense_cs/proxy.tld"%>
<cs:ftcs>
<proxy:tojson store="${store}" total="${total}" />
</cs:ftcs>
```

56.3.4.4 Testing Custom Search

After the JSON is scripted, test the search function to ensure everything works properly. To test the search:

In the Contributor interface, from the search box, select **ProxyTest** as the search type. Click the **Search** icon.

The **Search** tab opens displaying the associated data.

Figure 56-4   Search Results

The following figure shows the same content in Thumbnail view.

Figure 56-5   Search Results in Thumbnail View
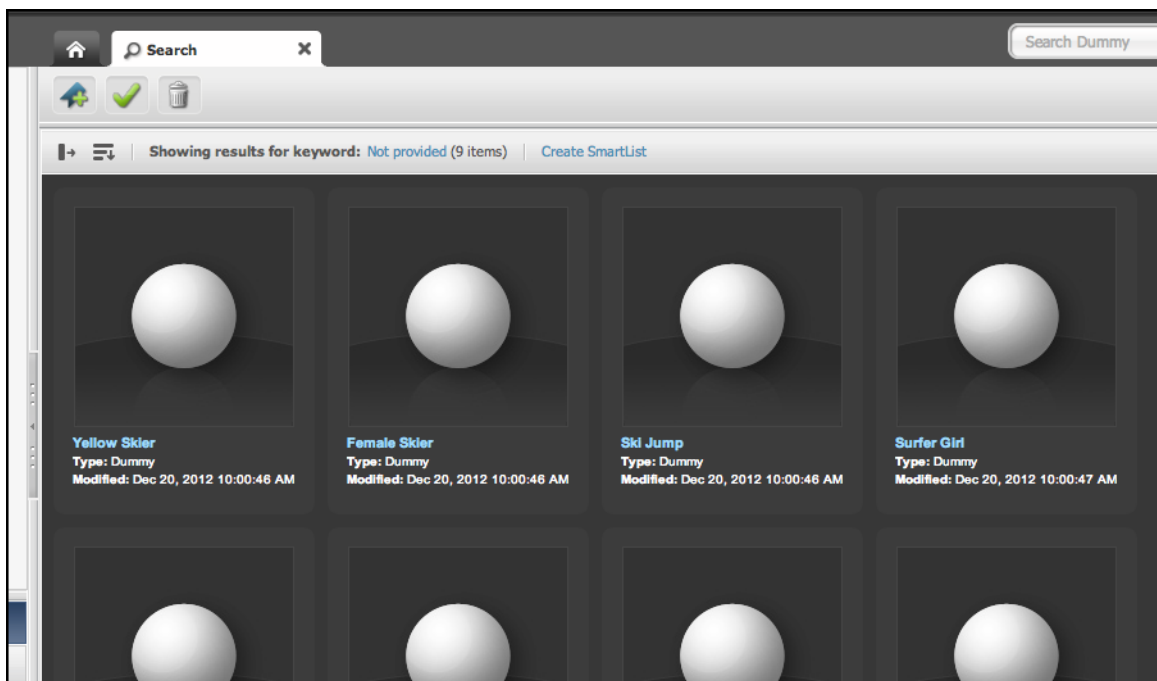
The **Search** tab also is functional in docked mode. The following figure shows that the tooltip is filled in with default data.

Figure 56-6   Search Results in Docked Panel
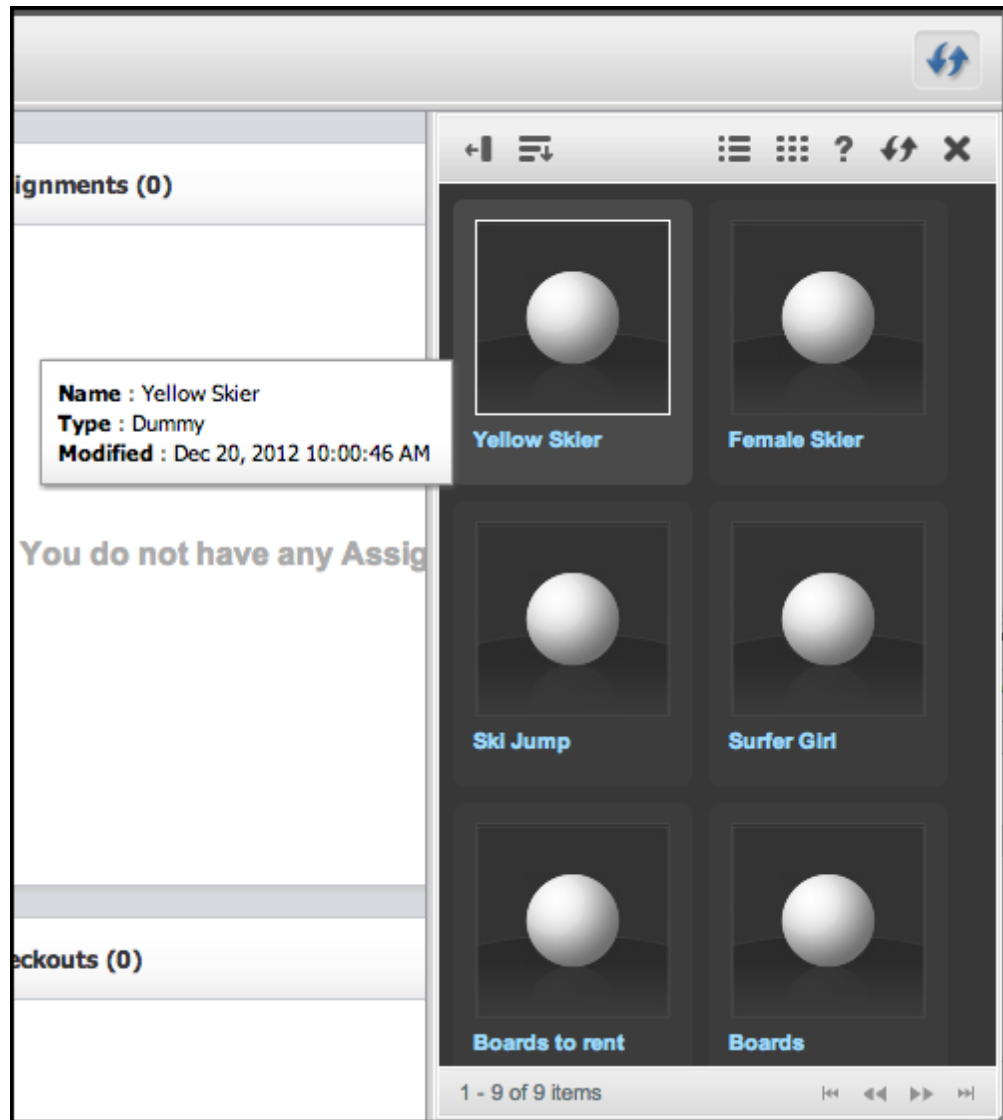
56.3.4.5 Additional Customizations

You can implement additional customizations. However, it is not necessary to use them to implement a custom search.

Rendering a Thumbnail

In the example, the ProxyTest repository returns a URL to a thumbnail image for each content item. We modify it to retrieve the thumbnail URL (sent by our ProxyTest repository) to render it. For that to happen, we must customize the grid in thumbnail view (whether docked and undocked); that is, overriding the configuration elements for:

```
UI/Layout/CenterPane/Search/View/ThumbnailView
UI/Layout/CenterPane/Search/View/DockedThumbnailView
```

Configuration files for each must be created:

```
CustomElements/ProxyTest/UI/Layout/CenterPane/Search/View/ThumbnailViewConfig
CustomElements/ProxyTest/UI/Layout/CenterPane/Search/View/DockedThumbnailViewConfig
```

The `ThumbnailViewConfig` configuration file has the following XML configuration:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld" %>
<%@ taglib prefix="xlat" uri="futuretense_cs/xlat.tld" %>
<cs:ftcs>
  <thumbnailviewconfig>
    <formatter>fw.ui.GridFormatter.simpleThumbnailFormatter</formatter>
  </thumbnailviewconfig>
</cs:ftcs>
```

You can see that this overrides the formatter setting for the ProxyTest thumbnail view (any other setting contained in the global configuration element is maintained). This formatter renders a default view for each search result showing a thumbnail image, and the **name** field (**inspect** link).

Similarly, docked thumbnail view settings should be overwritten by creating `DockedThumbnailViewConfig` in `CustomElements/ProxyTest/UI/Layout/CenterPane/Search/View/` with the following XML:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"%>
<%@ taglib prefix="ics" uri="futuretense_cs/ics.tld"%>
<cs:ftcs>
<ics:callelement element=
"[CustomElements/ProxyTest/UI/Layout/CenterPane/Search/View/ThumbnailViewConfig]" />
</cs:ftcs>
```

> **Note:**
>
> See Customizing Search Views of the Contributor Interface.
>
> The brackets around the element name indicate that the `ics:callelement` tag should not search for a customized version of the given element name.

Configuring the Grid Context Menu

Not all asset operations apply to proxy assets. Therefore, we have to override the default grid context menu setup. Override the element `UI/Config/GlobalHtml` by creating new element `MyConfig` in `UI/Config`. For more information about creating MyConfig, see Customizing Context Menus.

This ensures that none of the global settings are merged with the overridden settings.

Sorting Proxy Assets by Fields

Sorting of proxy assets is done by the external content source API (WebCenter Sites does not sort results internally). Thus, there may or may not be any support for sorting all fields. A user can carry out a sorted search for proxy assets by clicking the Sort icon and choosing among the sort options.

> **Note:** The `ProxyTest` asset example does not support sorting.
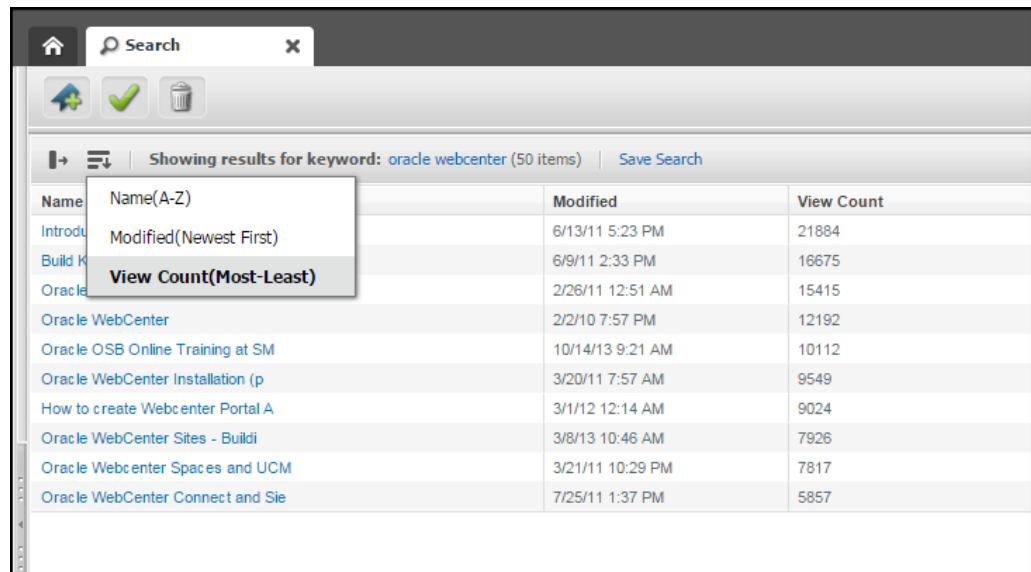
The sort options are set in `CustomElements/ProxyTest/UI/Layout/CenterPane/Search/View/SearchTopBarConfig`. You must define the following for the sort options:

- `fieldname`: The value of the `sort` variable that is passed with the request object to `SearchAction`, which should indicate the field to sort on (e.g. `date`).

- `displayname`: The string displayed in the **Sort** drop-down menu (e.g. `View Count(Most-Least)`).

- `sortorder`: `descending` or `ascending`. If `descending`, WebCenter Sites adds a negative sign (`-`) in front of the `fieldname` value.

The following code is written for the YouTube proxy assets:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"
%><%@ taglib prefix="xlat" uri="futuretense_cs/xlat.tld"
%><cs:ftcs>
    <!-- Here the field name is the index column on which the sort is performed.-->
    <sortconfig>
        <sortfields>
            <sortfield id="title">
                <fieldname>title</fieldname>
                <displayname><xlat:stream key="UI/UC1/Layout/NameSort1"
escape="true"/></displayname>
                <sortorder>ascending</sortorder>
            </sortfield>
            <sortfield id="updateddate_dsc">
                <fieldname>date</fieldname>
                <displayname><xlat:stream key="UI/UC1/Layout/ModifiedSort1"
escape="true"/></displayname>
                <sortorder>descending</sortorder>
            </sortfield>
            <sortfield id="viewCount">
                <fieldname>viewCount</fieldname>
                <displayname><xlat:stream key="UI/UC1/Layout/ViewCountSort"
escape="true"/></displayname>
                <sortorder>descending</sortorder>
            </sortfield>
        </sortfields>
```

```
        </sortconfig>
</cs:ftcs>
```

After a sort field is chosen, a new sorted search is initiated and `SearchAction` is called. A sort parameter is available in the `request` object, which provides the field to sort by. For example, in the YouTube proxy asset `SearchAction`, the sort parameter is obtained and used as a query parameter value:

```
...

String sort = request.getParameter("sort");
if(sort != null && StringUtils.isNotEmpty(sort)){
    // incase of descending sort, UI sends the field with a -ve sign,
    // YouTube doesn't take as is, so remove the -ve sign if any
    sort.trim();
    if(sort.startsWith("-")){
        sort = sort.substring(1,sort.length());
    }
}else{
    sort = "relevance";
}

...

// build YouTube URL
String baseYtURL = "https://www.googleapis.com/youtube/v3/";
StringBuffer ytQuery = new StringBuffer(baseYtURL);
ytQuery.append("search?part=snippet");
ytQuery.append("&maxResults=" + maxResults); // number of results per page
ytQuery.append("&type=video"); //only return videos, no channels or playlists
ytQuery.append("&order=" + sort); // ordering
ytQuery.append("&key=" + apiKey); // add user's API public access key
ytQuery.append("&q=").append(URLEncoder.encode(searchTerm));

...
```

Be aware that both `SearchTopBarConfig` and `ContextMenuConfig` elements must have `merge=false` set in the element `resdetails1` (or `resdetails2`) field.

Figure 56-7   Configuration File Element Details Value Field

elementname	descripti...	url	resdetails1	res
● ContextMenuConfig		CustomElements/Dummy/UI/Layout/CenterPane...	merge=false	
● DockedThumbnailViewConfig		CustomElements/Dummy/UI/Layout/CenterPane...		
● SearchTopBarConfig		CustomElements/Dummy/UI/Layout/CenterPane...	merge=false	
● ThumbnailViewConfig		CustomElements/Dummy/UI/Layout/CenterPane...		

56.3.5 Implementing a Custom Tree

A custom tree is useful to allow users to browse external content by category.

Figure 56-8   Custom Tree
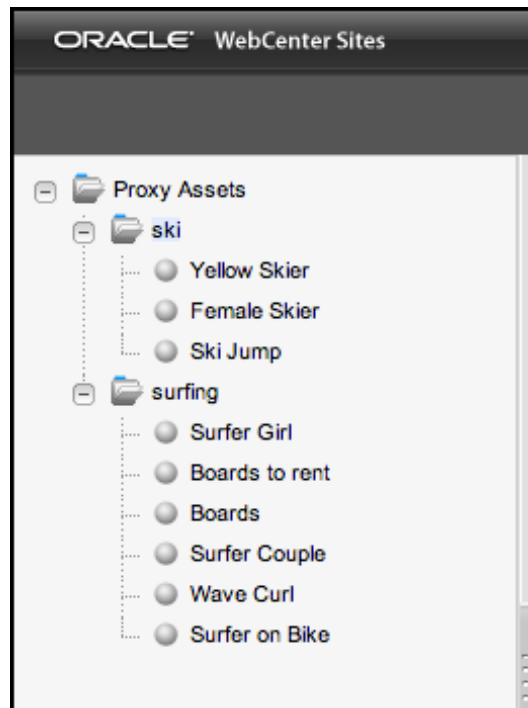
Two elements are created to render the tree:

- `ProxyTest/Tree/Root`: Renders the tree root nodes, that is, the content categories.

- `ProxyTest/Tree/Load`: Renders all content under a given category.

To implement the custom tree, you must first register the custom tree tab, and then implement the custom tree code.

This topic includes the following:

- Registering the Custom Tree Tab

- Implementing the Tree Code

56.3.5.1 Registering the Custom Tree Tab

The custom tree tab is defined in the Add New Tree Tab page. With the tab created in this example, a new content tree called Proxy Assets is created, containing a single custom section (ProxyTest) pointing at `ProxyTest/Tree/Root`.

To register the custom tree tab for this example:

1. In the Admin interface, expand the **Admin** node, then double-click **Tree**.

   The Tree Tabs page opens.

2. At the bottom of the Tree Tabs page, click **Add New Tree Tab**.

   The Add New Tree Tab page opens.

3. Fill in the fields as needed. For this specific example, fill in the fields in this way:

   - **Title**: Enter `Proxy Assets` for the title of the tab.

- **Sites**: Select **Proxy** from the list.

- **Required Roles**: Select **Any** from the list.

- **Tab Contents**: Select **ProxyTest** and click **Add Selected Items** to move it to the Selected column. **ProxyTest** should be the only item in the selected column.

- **Section Name**: Enter `ProxyTest` in the field.

- **Element Name**: Enter `ProxyTest/Tree/Root` in the field.

Figure 56-9   Add New Tree Tab Page

4. Once the fields have been properly entered, click **Save** to save the asset.

56.3.5.2 Implementing the Tree Code

1. In `ProxyTest/Tree/Root`, query the external repository to get the tree root nodes (in this case, the list of categories).

2. Generate the Tree node data for each category:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"%>
<%@ taglib prefix="satellite" uri="futuretense_cs/satellite.tld"%>
<%@ taglib prefix="ics" uri="futuretense_cs/ics.tld"%>
<%@ page import="org.codehaus.jettison.json.*"%>
<cs:ftcs>

<%-- Retrieve all categories from ProxyTest repository --%>
<ics:setvar name="serviceURL" value="/browse/categories.json" />
<ics:callelement element="ProxyTest/GetData" />

<%
JSONArray list = (JSONArray)ics.GetObj("items");
for (int i = 0; i < list.length(); i++) {
     String category = (String)list.get(i);
     // 1) build a tree node id - needs to be unique
     // Note: we need to make sure that AssetType is not present in the scope
     // (BuildTreeNodeID is otherwise assuming the tree node to be an asset node)
     %>
     <ics:removevar name="AssetType" />
     <ics:callelement element="OpenMarket/Gator/UIFramework/BuildTreeNodeID" >
          <ics:argument name="AdHoc" value='<%=category %>' />
     </ics:callelement>
     <%
     // 2) build the 'LoadURL' that is, the URL to call to load this node's
children
     // OpenMarket/Gator/UIFramework/LoadTab is a standard element.
     // Required input is:
     // - populate: the element rendering tree nodes to execute
     // - op: must be set to 'load'
     //
     // We're also passing 'category' which is required by our custom logic in
     // ProxyTest/Tree/Load
     //
     %>
     <satellite:link assembler="query"
                         pagename="OpenMarket/Gator/UIFramework/LoadTab"
                         outstring="LoadURL">
     <satellite:argument name="populate" value="ProxyTest/Tree/Load"/>
     <satellite:argument name="op" value="load"/>
     <satellite:argument name="category" value='<%=category %>' />
     </satellite:link>

     <ics:callelement element="OpenMarket/Gator/UIFramework/BuildTreeNode">
          <ics:argument name="Label" value='<%=category %>' />
     </ics:callelement>
<%}%>
</cs:ftcs>
```

This figure shows the tree generated with this code:

Figure 56-10   Generated Custom Tree
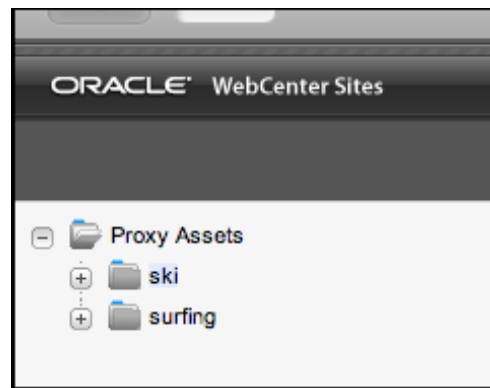
> **Note:**
>
> The elements `OpenMarket/Gator/UIFramework/BuildTreeNodeID` and
> `OpenMarket/Gator/UIFramework/BuildTreeNode` both consume
> variables in the ICS scope. Since the `<ics:callelement />` tag has a global
> scope, it is, in some cases, necessary to explicitly remove (permanently or
> temporarily) certain variables from the ICS scope, using
> `<ics:removevar />`.

3. In `ProxyTest/Tree/Load`, query the external service to retrieve all content in a
   given category, then render a node for each content item:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"%>
<%@ taglib prefix="ics" uri="futuretense_cs/ics.tld"%>
<%@ taglib prefix="proxy" uri="futuretense_cs/proxy.tld"%>
<%@ page import="org.codehaus.jettison.json.*"%>
<cs:ftcs>

<%-- Retrieve all content for a given category --%>
<ics:setvar name="serviceURL" value='<%="/browse/" + ics.GetVar("category") +
    ".json"%>' />
<ics:callelement element="ProxyTest/GetData" />

<%
JSONArray list = (JSONArray)ics.GetObj("items");

for (int i = 0; i < list.length(); i++) {
    JSONObject item = (JSONObject)list.get(i);%>

    <proxy:register externalid='<%=item.getString("id") %>'
                    type="ProxyTest"
                    name='<%=item.getString("title") %>'
                    varname="internalid" />

    <ics:callelement element="OpenMarket/Gator/UIFramework/BuildTreeNodeID">
       <ics:argument name="AssetType" value="ProxyTest" />
       <ics:argument name="ID" value='<%=ics.GetVar("internalid") %>' />
    </ics:callelement>

    <ics:callelement element="OpenMarket/Gator/UIFramework/BuildTreeNode">
       <ics:argument name="Label" value='<%=item.getString("title") %>' />
       <ics:argument name="Description" value='<%=item.getString("title") %>' />
       <ics:argument name="executeFunction" value="inspect" />
```

```
        </ics:callelement>
        <%
}
%>
</cs:ftcs>
```

You then should be able to browse each category.

Figure 56-11   Browsing Custom Tree Objects
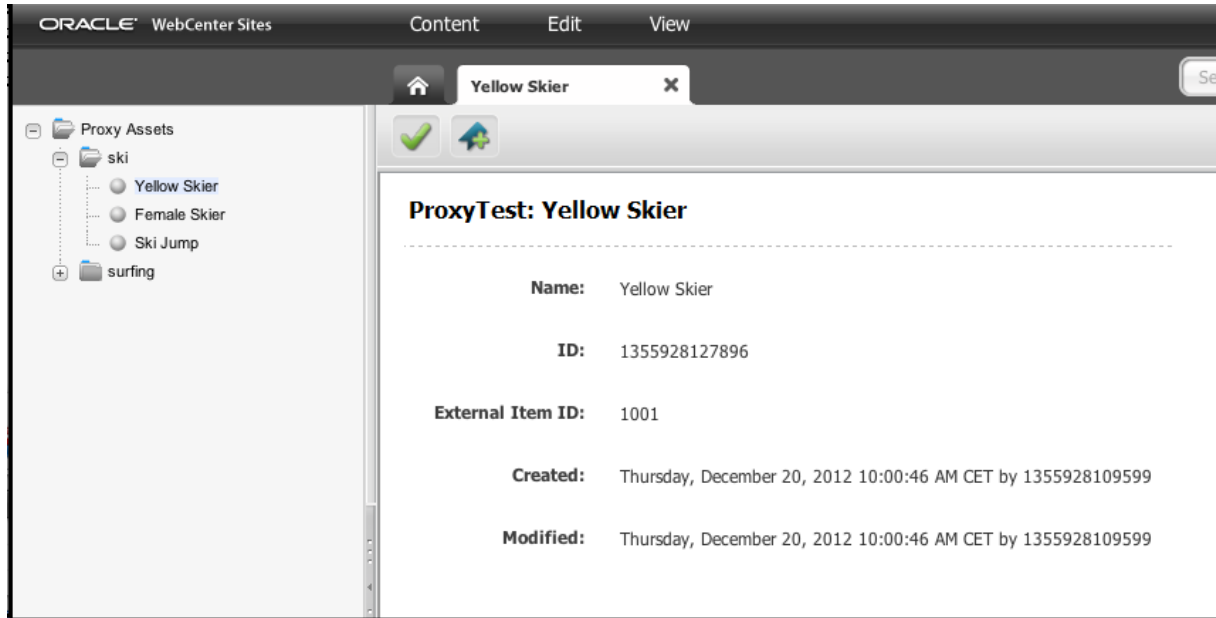
4. Double-click a node to open the default proxy asset inspection page for that node.

56.4 Setting Up YouTube Proxy Assets

To access YouTube proxy assets, you need to add an API key and set up a proxy server or WebLogic Server– if your WebCenter Sites instance is installed on WebLogic Server.

To set up YouTube proxy assets:

1. Obtain the Google Developers YouTube API key:

    a. Register your project in the Google Developers Console.

    b. Enable the YouTube Data API v3.

    c. Generate a public API server key.

2. Copy the generated key into the properties file:

    a. Open the `wcs_properties.json` file.

    b. Search for the `wcsites.youtube.api.key` key, and add the generated API key into the value:

    ```
    {
    "key" : "wcsites.youtube.api.key",
    "value" : "AIzaSyBnYqexNhlvU6QaJ2emS829bF...",
    "valid_values" : [ "" ],
    ```

```
"defaultValue" : "",
"category" : "Core",
"subcategory" : "",
"global" : true,
"localServerValues" : { },
"hide" : false,
"readonly" : false,
"restart_required" : false,
"deprecated" : false,
"description" : "YouTube public API access key used to search Youtube videos
(proxy assets). Created by registering project in the Google Developers
Console, enabling YouTube Data API v3, and generating a public API server key."
}
```

3. Search and use the YouTube proxy asset that make use of the YouTube API v3:

   a. Switch to the avisports site.

   b. Go to the Contributor interface.

   c. In the Search bar, set the search type as YouTube Video.

   d. Enter a search term and click the **Search** icon.

      YouTube videos are displayed.

4. If you are using a proxy server, follow these steps to configure it for YouTube proxy assets:

   a. In Sites Explorer, navigate to **Tables** > **ElementCatalog** > **CustomElements** > **YouTube** > **UI** > **Data** > **Search**.

   b. Edit the row for **SearchAction.jsp**.

      • In the **resdetails1** column, enter the proxy host. For example, `proxyHost=my-proxy.company.com`.

      • In the **resdetails2** column, enter the proxy port. For example, `proxyPort=80`.

5. If you are using the WebLogic Server on which WebCenter Sites is installed, follow these steps to configure the WebLogic Server for YouTube proxy assets:

   a. Open the Admin console: `host:admin port/console`.

   b. Expand **Domain Structure**, then **Environment**, and then **Servers**.

   c. Click the managed server on whichWebCenter Sites is running.

   d. Under the **SSL** tab, expand **Advanced**.

      • Under **Hostname Verification**, select **Custom Hostname Verifier**.

      • For **Custom Hostname Verifier**, enter `weblogic.security.utils.SSLWLSWildcardHostnameVerifier`.

   e. Ensure that the WebLogic server's truststore trusts `googleapis.com`. WebLogic's default settings allow this:

      • Under the **Keystores** tab, for **Keystores**, click **Change**, and select **Demo Identity and Demo Trust**.

- Under the **Keystores** tab, ensure that the **passphrase** contains the Java Standard Trust Keystore (default is `changeit`).

6. Restart the server.

56.5 User Interface Customizations

You can make WebCenter Sites an enjoyable experience for contributors and marketers if you customize the Contributor interface in ways that make the common tasks easier. For instance, you may want to customize search to help them search content faster.

This topic describes customizable portions of the interface to synchronize with third-party software.

- Customizing the Search Start Menu

- Customizing the Content Tree

56.5.1 Customizing the Search Start Menu

In the Contributor interface, users run searches restricted to a specific asset type by selecting a **Search** start menu. This figure shows the **Search Type** list:

Figure 56-12   Search Type Start Menu Selection
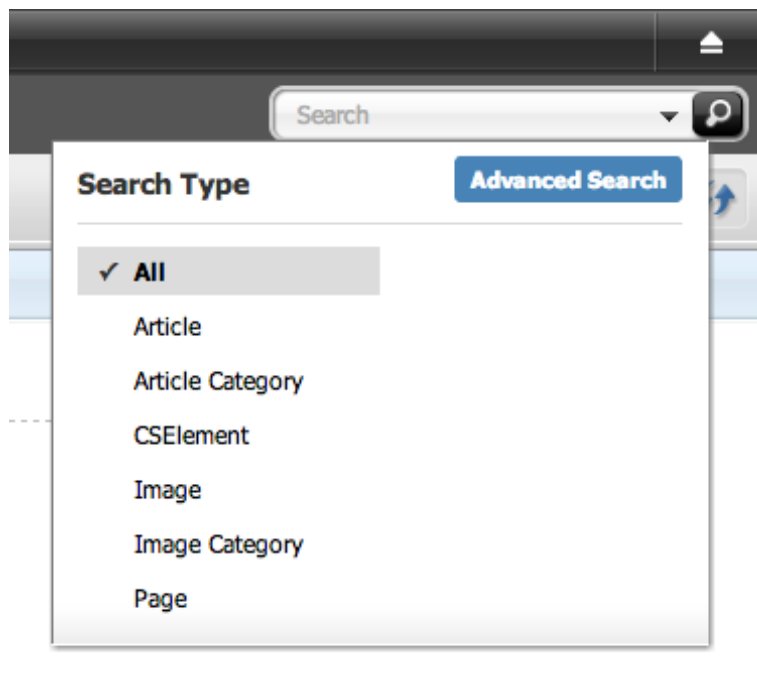
To customize search for a given asset type, override the controller element `UI/Data/Search/Search` by creating the following elements:

```
CustomElements/<AssetType>/UI/Data/Search/SearchAction
CustomElements/<AssetType>/UI/Data/Search/SearchJson
```

For more details about controller elements and element overrides, see Customizing Search Views of the Contributor Interface.

UI/Data/Search/Search runs the search code and generates the appropriate JSON data for the grid widget's consumption (whether in list or thumbnail view, docked or undocked). The JSON data must be a valid Dojo datastore. For an implementation example, see Customizing the Content Tree.

56.5.2 Customizing the Content Tree

The content tree can be customized by defining a custom tree section, contained in a new or existing tree tab: For more details about defining custom tree tab sections, see Options for Managing Access to the Tree (Admin Interface Only).

A custom tree tab section points to a JSP element generating data based on some rendering logic, eventually consumed by the tree widget. Tree data is generated using the following utility elements:

- OpenMarket/Gator/UIFramework/BuildTreeNodeId: Generates a tree node ID.

- OpenMarket/Gator/UIFramework/BuildTreeNode: Generates properly formatted tree node data.

The following examples show how to build a tree node, whether it represents an asset or not (that is, an asset node compared to an *adhoc* node). See Customizing the Tree in the Admin Interface.

Example 1: Build a Tree Node Representing an Asset

This example shows how to generate a tree node representing a single asset node. The node runs the inspect action when double-clicked (see executeFunction parameter); that is, the asset default inspect view is rendered in a new tab (or focused if the asset is opened in a tab).

```
<%--
- Generates a tree node id.
- The element expects the asset id and type to be passed in parameters
- called respectively "ID" and "AssetType".
- The generated id is stored in a WebCenter
Sites variable called "TreeNodeID"
--%>
<ics:callelement element="OpenMarket/Gator/UIFramework/BuildTreeNodeID">
     <ics:argument name="AssetType" value="Article" />
     <ics:argument name="ID" value="1234567890" />
</ics:callelement>

<%--
- Generates a tree node for this asset.
- Note that this element implicitly consumes variables
- currently present in the ICS scope, including "TreeNodeID"
--%>
<ics:callelement element="OpenMarket/Gator/UIFramework/BuildTreeNode">
     <ics:argument name="Label" value="Node Label, for example asset name or other
                  readable string" />
     <ics:argument name="Description" value="Some optional tooltip text" />
     <ics:argument name="executeFunction" value="inspect" />
</ics:callelement>
```

Example 2: Build an adhoc Tree Node

Tree nodes do not necessarily represent assets, in which case they are called *adhoc* nodes. This example shows how to generate an *adhoc* node representing a parent node, a node that has children and can be expanded and collapsed.

```
<%--
- Generates a tree node id.
- For adhoc nodes, the expected parameter is called "AdHoc",
- and can be any arbitrary string, which must be unique across tree nodes
--%>
<ics:callelement element="OpenMarket/Gator/UIFramework/BuildTreeNodeID" >
  <ics:argument name="AdHoc" value="SomeUniqueString" />
</ics:callelement>

<%--
- Generates a "LoadURL", that is the URL to be called when a tree node
- is expanded. In this case, we're calling the default utility SiteCatalog entry
- (OpenMarket/Gator/UIFramework/LoadTab) which, in turn, will invoke
- a custom element ("Some/Other/Element" in our example), in charge
- of generating the child nodes, based on some custom logic.
--%>
<satellite:link
        assembler="query"
        pagename="OpenMarket/Gator/UIFramework/LoadTab"
        outstring="LoadURL">
  <satellite:argument name="populate" value="Some/Other/Element"/>
  <satellite:argument name="op" value="load"/>
</satellite:link>

<%--
- Generates the corresponding tree node data
- Note that this element consumes the "LoadURL" variable previously generated.
- The presence of LoadURL indicates whether a node should be marked as expandable
or not.
--%>
<ics:callelement element="OpenMarket/Gator/UIFramework/BuildTreeNode">
  <ics:argument name="Label" value="Some meaningful node label" />
</ics:callelement>
```

56.6 Information About Embedding Proxy Assets in Web Pages

You can define templates for proxy assets to make them appealing on the website and embed proxy assets inside pages.

Topics:

- [Writing a Template for Proxy Assets](#)

- [Using Proxy Assets in Slots](#)

- [About Caching Proxy Assets](#)

56.6.1 Writing a Template for Proxy Assets

Templates can be defined for proxy assets, just as templates can be defined for any other asset type.

Using a Template to Render Proxy Assets:

1. In the Admin interface, click **New** on the menu bar and then choose **New Template**.

   The Template form is displayed.

2. In the **Name** field, enter `ProxyTestSummary`.

3. From the **For Asset Type** drop-down list, choose **ProxyTest**.

4. In the **Applied to subtypes** box, choose **Any**, if not selected already.

5. On the **Element** tab:

- From the **Usage** drop-down list, choose **Element is used as Layout**.

- In the **Create Template Element?** section, click **JSP** .

- In the **Element Logic** box, enter the following code from `${WCS_PROXY}`
  `\templates\ProxyTestSummary.jsp`:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"
%><%@ taglib prefix="ics" uri="futuretense_cs/ics.tld"
%><%@ taglib prefix="render" uri="futuretense_cs/render.tld"
%><%@ taglib prefix="fragment" uri="futuretense_cs/fragment.tld"
%><%@ page import="javax.ws.rs.client.*"
%><%@ page import="javax.ws.rs.core.*"
%><%@ page import="org.codehaus.jettison.json.*"
><%@ page import="com.fatwire.cs.ui.framework.UIException"
%><%@ taglib prefix="asset" uri="futuretense_cs/asset.tld"%>

<cs:ftcs>
<%-- ProxyTest/ProxyTestSummary    --%>

<%-- Record dependencies for the Template --%>
<ics:if condition='<%=ics.GetVar("tid")!=null%>'>
    <ics:then><render:logdep cid='<%=ics.GetVar("tid")%>' c="Template"/>
    </ics:then>
</ics:if>

<%-- Set the urlPath to proxy_samples --%>
<%
String host = request.getServerName();
String port = Integer.toString(request.getServerPort());
String contextPath = request.getContextPath();
String urlPath = "http://"+host+":"+port+"/"+contextPath+"/proxy_samples/
content/";
%>

<%-- first, retrieve the external id --%>
<asset:load name="asset" type='<%=ics.GetVar("c") %>'
       objectid='<%=ics.GetVar("cid") %>' />
<asset:get name="asset" field="name" />
<asset:get name="asset" field="externalid" />

<%
// given the external content id, invoke the  third-party repository API
// to retrieve content metadata
Client client = ClientBuilder.newClient();
Response resp = null;
WebTarget res = null;

try {
    res = client.target(urlPath + ics.GetVar("externalid") + ".json");
    resp = res.request(MediaType.APPLICATION_JSON).get();
} catch (Exception e) {
    // propagate exception to client-side
    request.setAttribute(UIException._UI_EXCEPTION_, e);
    e.printStackTrace();
```

```
        throw e;
    }

    String jsonString = resp.readEntity(String.class);
    JSONObject data = new JSONObject(jsonString);
    %>

    <%-- finally render content --%>
    <h4><%=data.getString("title") %></h4>
    <p><%=data.getString("foo") %></p>
    <img src="<%=data.getString("image") %>" alt="<%=data.getString("title") %>" />

    </cs:ftcs>
```

6. In the Contributor interface, search for `ProxyTest` assets while leaving the **Search** field blank.

7. Click a `ProxyTest` asset that includes asset details (for example, **YellowSkier**).

8. Click **Preview** , then choose the **ProxyTestSummary** layout, and then click **Apply**.

Title, metadata, and image of the `ProxyTest` asset should be displayed.
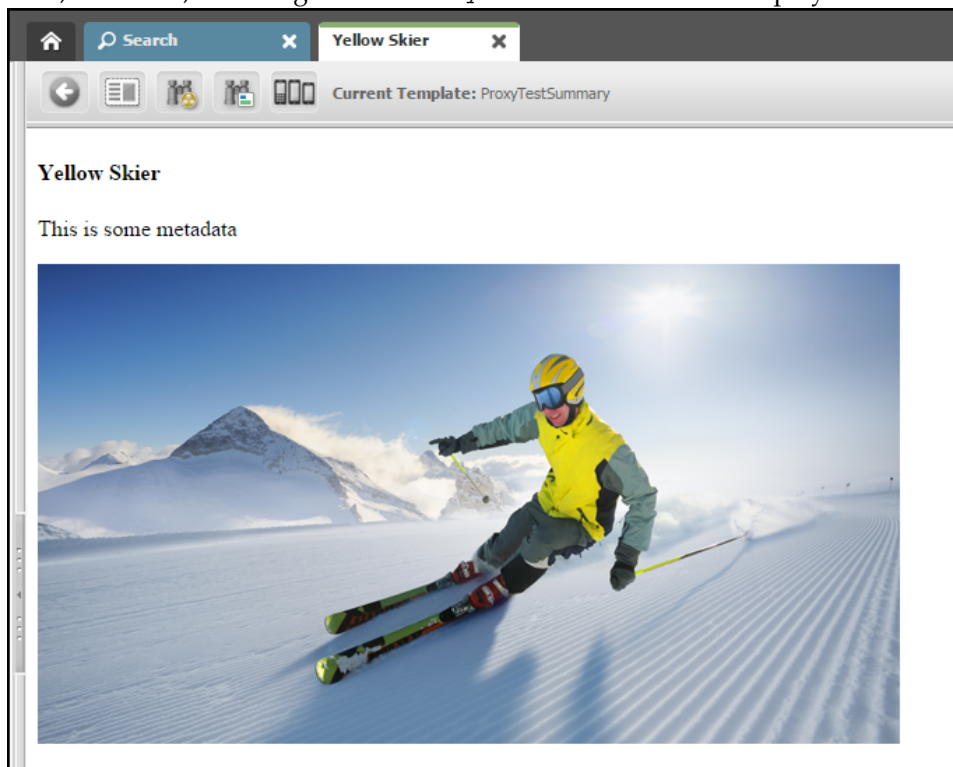
56.6.2 Using Proxy Assets in Slots

Proxy assets can be embedded inside pages using the exact same tags and principles that apply to standard assets.

1. Create a Page attribute:

   a. In the **Name** field, enter `ProxyTestAttribute`.

   b. From the **Attribute Type** drop-down list, choose **asset**.

c. From the **Asset Type** drop-down list, choose **ProxyTest**.

d. Click the **Save** icon.

2. Create a Page definition:

a. In the **Name** field, enter `ProxyTestPageDef`.

b. In the **Attributes** section, select the **ProxyTest** attribute and then click **Optional**.

c. Click the **Save** icon.

3. Create a template for the Page asset:

a. In the **Name** field, enter `ProxyTestSlotTemplate`.

b. From the **For Asset Type** drop-down list, choose **Page**.

c. In the **Applied to subtypes** box, choose **Any**, if not selected already.

d. On the **Element** tab:

- From the **Usage** drop-down list, choose **Element is used as Layout**.

- In the **Create Template Element?** section, click **JSP** .

- In the **Element Logic** box, enter the following code from `${WCS_PROXY}`
  `\templates\ProxyTestSlotTemplate.jsp`

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"%>
<%@ taglib prefix="ics" uri="futuretense_cs/ics.tld"%>
<%@ taglib prefix="render" uri="futuretense_cs/render.tld"%>
<%@ taglib prefix="insite" uri="futuretense_cs/insite.tld"%>
<%@ taglib prefix="assetset" uri="futuretense_cs/assetset.tld"%>
<cs:ftcs><render:logdep cid='<%=ics.GetVar("tid")%>' c="Template"/>
<html>
<head>
     <title>ProxyTest Slots Example</title>
</head>
<body>
     <assetset:setasset name="page" id='<%=ics.GetVar("cid") %>'
                        type='<%=ics.GetVar("c") %>' />
     <assetset:getattributevalues attribute="ProxyTestAttribute"
listvarname="ProxyTest"
                        name="page" typename="PageAttribute" />
     <ics:listget listname="ProxyTest" fieldname="value" output="id" />
     <h3>ProxyTest Slots Example</h3>
     <insite:calltemplate field="ProxyTestAttribute" c='ProxyTest'
                        cid='<%=ics.GetVar("id") %>'
tname="ProxyTestSummary" />
</body>
</html>
</cs:ftcs>
```

4. Create the Page asset:

a. In the Contributor interface, from the menu bar, select **Content**, **New**, and then select **Page (Section)**.

b. In the **Name** field, enter **ProxyTestPage**.

c. From the **Page Definition** drop-down list, choose **ProxyTestPageDef**.

    d. From the **Template** drop-down list, choose **ProxyTestSlotTemplate**.

    e. In form mode and web mode, drag and drop `ProxyTest` assets into the **ProxyTestAttribute** slot.

    In web mode, the `ProxyTest` asset is rendered using the `ProxyTestSummary` template.
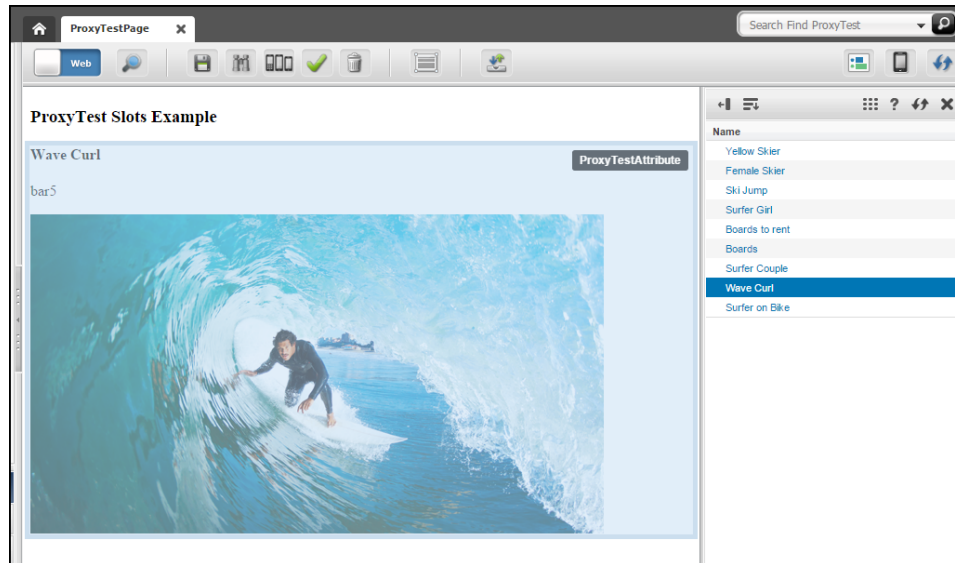
56.6.3 About Caching Proxy Assets

Because the rendered content is stored and managed in a separate repository, WebCenter Sites has no way to know when any content is modified, or even if content is modified. Oracle recommends to set cache expiration to a limited period of time on templates rendering proxy assets, such as the Summary template defined in Writing a Template for Proxy Assets.

Part XVII

Developing Applications with the Web Experience Management (WEM) Framework

Become familiar with the Web Experience Management (WEM) Framework that lets you develop applications. Learn the process of developing applications and custom Representational State Transfer (REST) resources. Learn how you can implement and customize Single Sign-On (SSO).

- About the Web Experience Management (WEM) Framework

- Understanding the WEM Framework and Services

- Working with the Articles Sample Application

- Developing Applications with WEM Framework

- Developing Custom REST Resources with WEM Framework

- Working with Single Sign-On for Production Sites

- Using REST Resources with the WEM Framework

- Introducing Customizable Single Sign-On Facility in WEM Framework

- Buffering in WEM Framework

- Registering Applications Manually in WEM Framework

57

About the Web Experience Management (WEM) Framework

With the Web Experience Management (WEM) Framework you can develop applications and integrate them with Oracle WebCenter Sites. Through its single administrative interface, WEM Admin, you can centrally manage applications and user authorization. WEM lets you implement single sign-on so the users log in only once to gain access to all applications that are allowed to them during a session.

Topics:

- About the WEM Framework

- Prerequisites for Application Development

- Getting Started

57.1 About the WEM Framework

The WEM Framework provides you the technology to develop and integrate applications with Oracle WebCenter Sites. The WEM Framework relies on WebCenter Sites for content management, and it's shipped with the WebCenter Sites Representational State Transfer (REST) API.

Objects in the WebCenter Sites database, such as sites, users, and data model map to REST resources in the WEM Framework.

Figure 57-1   WEM Framework
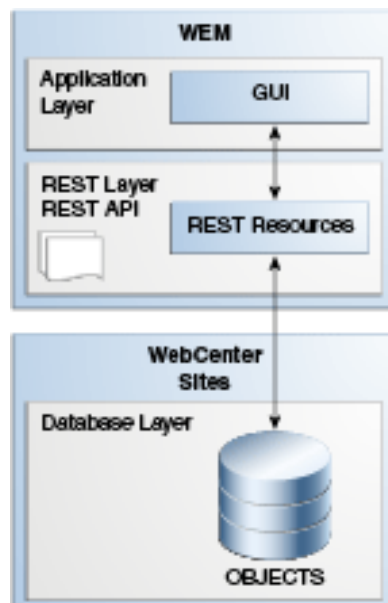
When implemented on the WEM Framework, applications communicate with the WebCenter Sites database through REST services. The applications appear in WEM Admin as list items on the Apps page. Administrators authorize users, which involves configuring access to the applications and their resources. To this end, the WEM Admin interface exposes authorization items (along with applications) through links on the menu bar.

Figure 57-2   Apps Page, WEM Admin
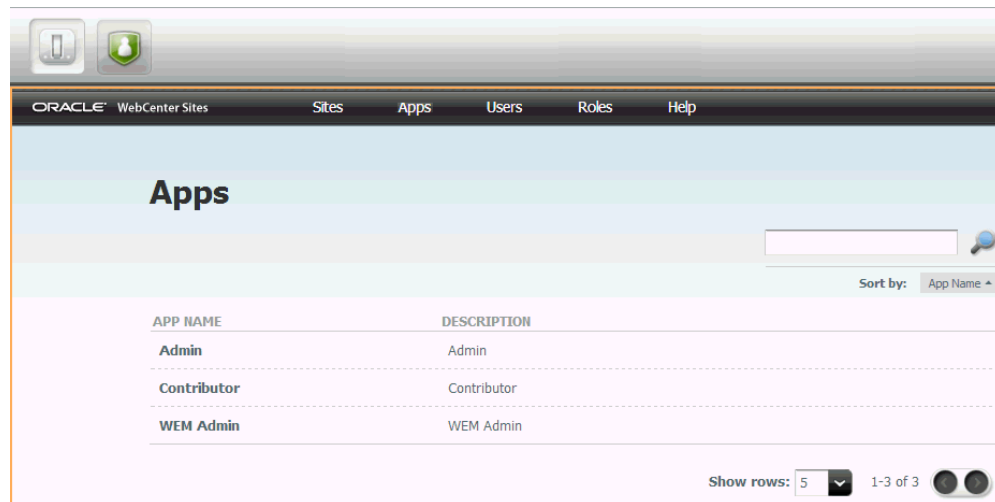
Coupling the items enables applications for users.

Figure 57-3   Authorization Model
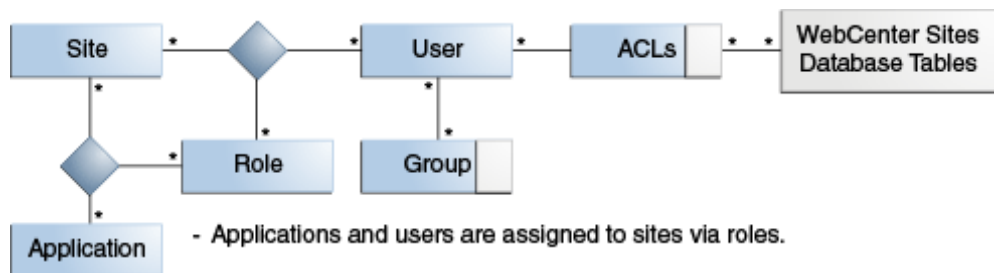
- Applications and users are assigned to sites via roles.

- Sharing a role to a user and an application on the same site grants the user access to the application on that site.

- Users are assigned to groups, which control access to applications' resources (REST resources).

- ACLs are assigned to users, providing them with access to the system.

- Using WEM Admin, general administrators can create and otherwise manage sites, applications, users, and roles. Groups and ACLs must be configured in the WebCenter Sites Admin interface. They are exposed in WEM Admin, in user accounts.

Once the coupling is complete, users are authorized at the database, REST, and application levels.

Figure 57-4   Roles, Groups, and ACLs
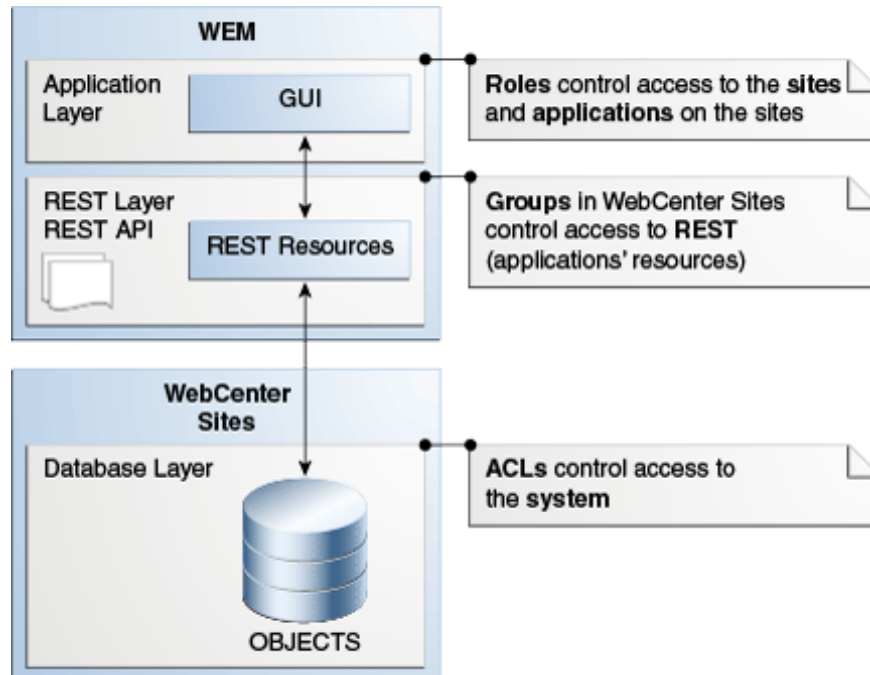
Experienced WebCenter Sites developers will recognize that the WEM Framework extends the use of sites and roles to control access to applications. However, unlike WebCenter Sites, the WEM Admin interface does not expose the data model. The REST API does. In this respect, WEM Admin can be thought of as strictly an authorization interface, supported by the Admin interface (for configuring ACLs and groups).

Although WEM Admin is seldom used by developers, the concepts behind user authorization can come into play in application development. The next chapters describe the WEM Framework as it relates to application development and provides examples of application code.

57.2 Prerequisites for Application Development

When you're developing an application with WEM, you code the application's logic, deploy the application, and register the application to expose it in WEM Admin interface. Administrators can manage the applicable from this interface and make it available to other users. You need to be an expert WebCenter Sites developers, and you should have a working knowledge of the technologies that are discussed for reference here.

Topics:

- Technologies

- WebCenter Sites Interfaces, Objects, and APIs

- Documentation

- Sample Applications and Files

- Application Access

57.2.1 Technologies

- Representational State Transfer (REST), used to communicate with the WebCenter Sites platform

- Central Authentication Service (CAS), which is deployed during WebCenter Sites installation to support single sign-on for WEM

- Java Server Pages Standard Tag Library (JSTL), Java, JavaScript, Jersey, and the Spring MVC framework, to follow the code of the Articles sample application provided with WEM

57.2.2 WebCenter Sites Interfaces, Objects, and APIs

Developers must have a working knowledge of:

- WebCenter Sites Admin (the administrative interface)

- WebCenter Sites basic and flex asset models

- Asset API

- ACLs, which protect database tables and define the types of operations that can be performed on the tables

- Concept of sites and roles

57.2.3 Documentation

This section discusses the following documents:

- *Oracle Fusion Middleware REST API Resource Reference for Oracle WebCenter Sites*

- *Oracle Fusion Middleware Java API Reference for Oracle WebCenter Sites*

Information about ACLs, sites, and roles, and their usage in WebCenter Sites is available in Working with ACLs and Roles in *Oracle Fusion Middleware Administering Oracle WebCenter Sites*.

57.2.4 Sample Applications and Files

- The following sample applications are used in this part of the guide:

  - *Articles*, a lightweight content management application

  - SSO sample application, a small authentication application for production sites. The application is packaged as `wem-sso-api-cas-sample.war`.

  - *Recommendations*, which demonstrates the process of creating REST resources

- The Customizable Single Sign-On facility is used in this section of the guide to illustrate customization of login behavior.

- WEM Framework ships with sample files to illustrate cross-domain implementations and management of assets over REST using our API.

All sample applications and files are located in the `Misc/Samples/WEM Samples` folder in your WebCenter Sites installation directory.

57.2.5 Application Access

When using information in this part of the guide, or developing and testing, access the **WEM Admin** interface to test the results of your application registration process as follows:

1. In a Web browser, access the following URL:

    ```
    http://wcs-server:wcs-port/wcs-context-root/
    ```

2. Log in as `fwadmin` (or an equivalent user).

3. In the page that appears, select **AdminSite**, then **Admin** (the first icon).

4. The

    WEM Admin **Sites** page appears. Registered applications are listed on the **Apps** page.

Figure 57-5   Sites Page
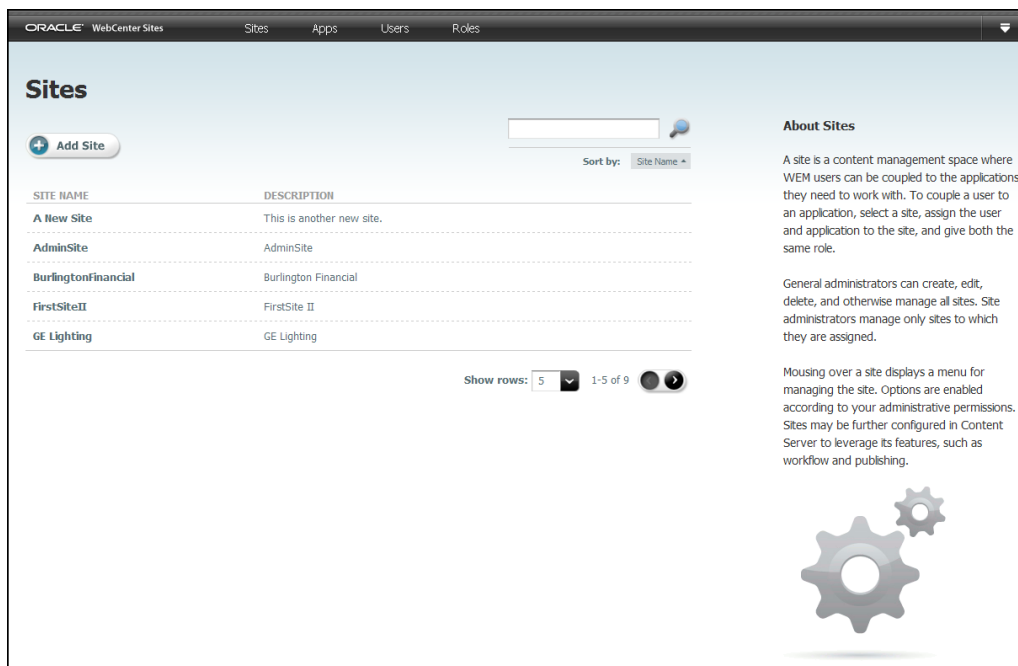
57.3 Getting Started

You'll work with a sample application called Articles to understand how you can use WEM to create applications. You'll learn about creating REST resources, single sign-on, and system security.

*   For information about the WEM Framework, see Understanding the WEM Framework and Services.

*   For a demonstration of the Articles application, see Working with the Articles Sample Application.

*   For information about the Articles application code, programmatic application registration, and cross-domain implementations, see Developing Applications

with WEM Framework. An example of manual application registration is available in Registering Applications Manually in WEM Framework.

- For information about creating REST resources, see Developing Custom REST Resources with WEM Framework.

- For a demonstration of the SSO sample application, see Working with Single Sign-On for Production Sites.

- For information about system security, see Using REST Resources with the WEM Framework.

- For information about customizing the login behavior for the WEM Framework, see Introducing Customizable Single Sign-On Facility in WEM Framework.

- For information about buffering, see Buffering in WEM Framework.

58

Understanding the WEM Framework and Services

The application developer's environment consists of the WEM Framework running on WebCenter Sites using REST services. Applications can be written in any language to make REST calls to WebCenter Sites. Custom-built applications can be deployed to an application server other than the platform's, and therefore written independently of the platform's deployment infrastructure.

For information about the WEM framework and services, see these topics:

- Support for Application Development

- REST Services

- UI Container

- Single Sign-On

- Authorization Model

- Custom Applications

- Requirements for REST Resources

58.1 Support for Application Development

When you're developing an application with WEM, you use REST services to access WebCenter Sites objects, UI container to expose applications, SSO to authenticate users, and REST authorization model to control access to REST resources.

Support for application development is in the following components (which are also described in their own sections in this chapter):

- **REST services,** a set of programmatic interfaces that provide access to the WebCenter Sites objects.

- **UI container,** which exposes registered applications. Registration enables rendering of the applications' interfaces. The UI container also supports the WEM Context object, used by applications to get details from the WEM Framework about the logged-in user and current site.

- **Single Sign-On (SSO),** which enables authenticated users to log in only once to access all applications allowed to them during the session. (The WebCenter Sites installation process installs the Central Authentication Service web application to support single sign-on in WEM.)

- **REST authorization model,** which provides fine-grained access control over REST resources, based on group membership. Application development does not

directly involve authorization (which is configured graphically in WEM Admin and the Admin interface), except when a predefined user is specified in the code.

WEM Admin is also part of the WEM Framework, but seldom used in application development, mainly to test the results of the application registration process, or to obtain administrative information about sites, users, groups, and roles. For information about WEM Admin and the Web Experience Management Framework, see *Oracle Fusion Middleware Administering Oracle WebCenter Sites*.

58.2 REST Services

With REST API, you can expose the WebCenter Sites data model and objects such as sites, users, roles, ACLs, groups, and so on.

- Basic asset types and basic assets (read-write)

- Flex asset types and definitions (read-only)

- Flex children and parents (read-write)

- Indexing to support asset searches

The following objects are also exposed by the REST API. They are used mainly by administrators in the authorization process (the objects are displayed in the WEM Admin interface):

- Sites (read-write)

- Users (read-write)

- Roles (read-write)

- ACLs (read-only)

- Groups (read-only), introduced in this release to control access to the REST layer.

- Auxiliary services: user locale and server time zone

(Sites, roles, and users can be configured in WEM Admin. ACLs and groups are exposed in WEM Admin (under Users) as read-only items; they must be configured in the Admin interface.)

Objects in WebCenter Sites map to REST resources in WEM. All other features, such as publishing, workflow, database management tools, and page caching must be accessed from the Admin interface or through JSP and XML tags.

Among the authorization objects that general administrators manage, sites and roles are the most likely candidates for application development, depending on your requirements. You can also specify predefined users to simplify administrators' authorization tasks.

- **Sites:** Using sites in application code is a requirement when the application's asset types and assets must be programmatically installed. The code must specify at least one site on which to enable the asset types (site-specific access to assets requires their asset types to be enabled on at least one site). Otherwise, install just the asset types (without naming any sites). Administrators will follow up by using the Admin interface to enable the asset types and assets on sites of their own choice.

- **Roles:** In WEM, roles are used to manage access to applications. Sharing a role to a user and an application on the same site grants the user access to the application

on that site. Roles can be used in application code to protect interface functions, such as `Edit`. The Admin interface exemplifies an application with role-protected interface functions.

- **Users:** The only user you are likely to specify in your application code is the predefined user, to simplify administrators' authorization processes. Specifying the user involves coding a user name and password. Instead of authorizing all application users individually at the REST level, an administrator will authorize your predefined user. Permissions granted to the predefined user will be passed to the logged-in users when they access the application. For more information about predefined users and the authorization model, see Authorization Model.

Keeping track of how sites and roles are used across the system is an administrator task that requires support from application developers. Tracking becomes especially important when the WebCenter Sites platform also functions as a staging system, only because the WEM Framework uses the WebCenter Sites database. For example, sites created in WEM Admin are stored in the database. They might not be used in WebCenter Sites for staging, but they are exposed in the Admin interface, along with its dedicated CM sites. Conversely, sites that are created in the Admin interface for CM purposes are exposed in WEM Admin, where other applications can be assigned to those sites. For users to be properly authorized, developers must communicate to administrators the nature of the custom-built applications: the resources they use, role-protected interface functions, and predefined users, if any.

58.3 UI Container

With the UI container, you expose registered applications in WEM Admin. Administrators manage these applications and make them available to other users. You use this container to support the Context object. These applications need this object to get details about the logged-in user and site (for example, the current site's name from the UI container) from the WEM Framework and to get utility methods that applications use to share data.

Topics:

- Registration

- WEM Context Object

58.3.1 Registration

The purpose of registering an application is to expose the application in WEM Admin for administrators to manage and make available to other users. Registration allows the system to recognize the application as an asset, which in turn allows the system to:

- List the application on the **Apps** page in WEM Admin

- Locate the icon you have chosen to represent the application

- Display the application's icon on the WebCenter Sites login page, and in the applications bar on each site to which the application is assigned

- Render the application's interface when the application's icon is selected

Figure 58-1   Registered Applications in UI Container
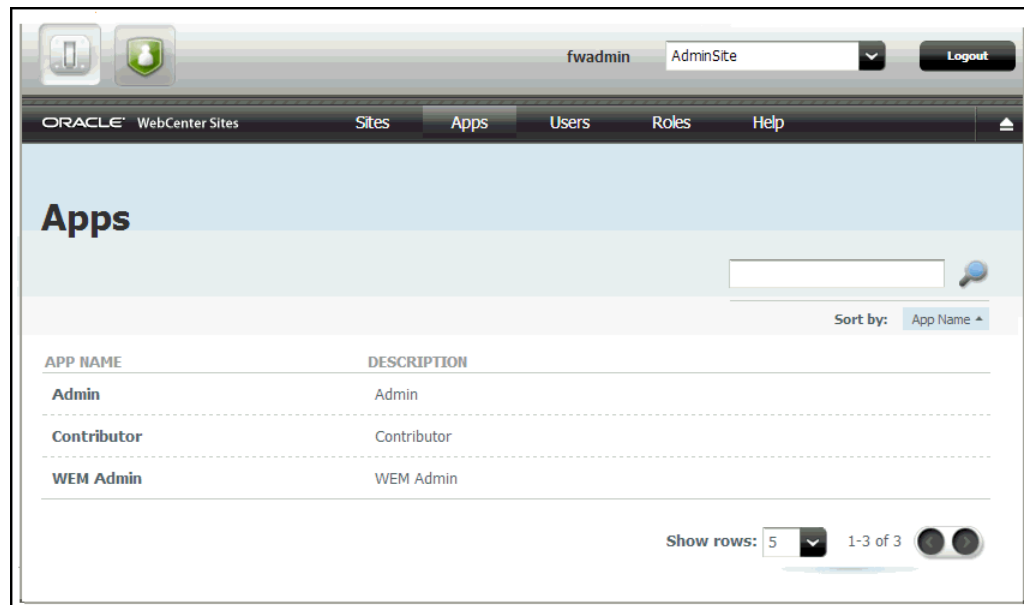
Registering an application includes registering its views. While multiple and shared views are supported, applications with a single, unshared view are typical (and used in this part of the guide). Views can be of type iframe, HTML, and JavaScript.

To support registration, the WEM Framework ships with the basic asset types `FW_Application` and `FW_View`. Both are created when the WEM option is selected during the WebCenter Sites installation process. They are enabled by default on AdminSite (also created during the WebCenter Sites installation process).

Registering an application (once it is deployed) requires creating an instance of `FW_Application`, creating an instance of `FW_View` for each view, and associating the `FW_View` instances with the `FW_Application` instance. Applications must be registered on AdminSite, even if they will be used on other sites. Registration allows applications to be assigned to other sites.

Applications can be registered either programmatically through the REST API's `applications` service, or manually from the Admin interface. Programmatic registration is preferred. For general instructions, see Registering Applications with Different Views. An example of manual registration is available in Registering Applications Manually in WEM Framework.

58.3.2 WEM Context Object

The UI container provides a JavaScript Context object (`WemContext`) to all applications inside the container. The Context object is used by the applications to get details from the WEM Framework about the logged-in user and site (for example, the current site's name from the UI container). The Context object also provides various utility methods that applications will use to share data. The Context Object can be used by applications running in the same domain as WebCenter Sites or in different domains. See Accessing Parameters from the WEM Framework.

58.4 Single Sign-On

You'll implement single-sign on with Central Authentication Service (CAS) which is a single sign-on protocol for the web. CAS permits a user to access multiple applications just by logging in only once.

You can read about Central Authentication Service here: http://www.jasig.org/cas. As shown in the sample Articles example, the servlet filter that ships with the WEM Framework is ready-to-use for any application that is deployed as a Java web application. If your application is developed using a different technology, refer to CAS clients specific to your choice of technology, at the following URL:

http://www.ja-sig.org/wiki/display/CASC/Official+Clients

When a user tries to access an application protected by CAS, the authentication system responds with the steps below.

1. **Initial Access**

   a. When the user first attempts to access an application protected by CAS, the user is redirected to the CAS login page.

   b. Upon successful login, the user is redirected back to the application with a ticket. The cookie for the CAS login page is saved.

   c. The application verifies the user's identity by verifying the ticket against CAS. (On content management systems, CAS authenticates by default against the WebCenter Sites database.)

   Figure 58-2   Accessing an Application Protected by CAS
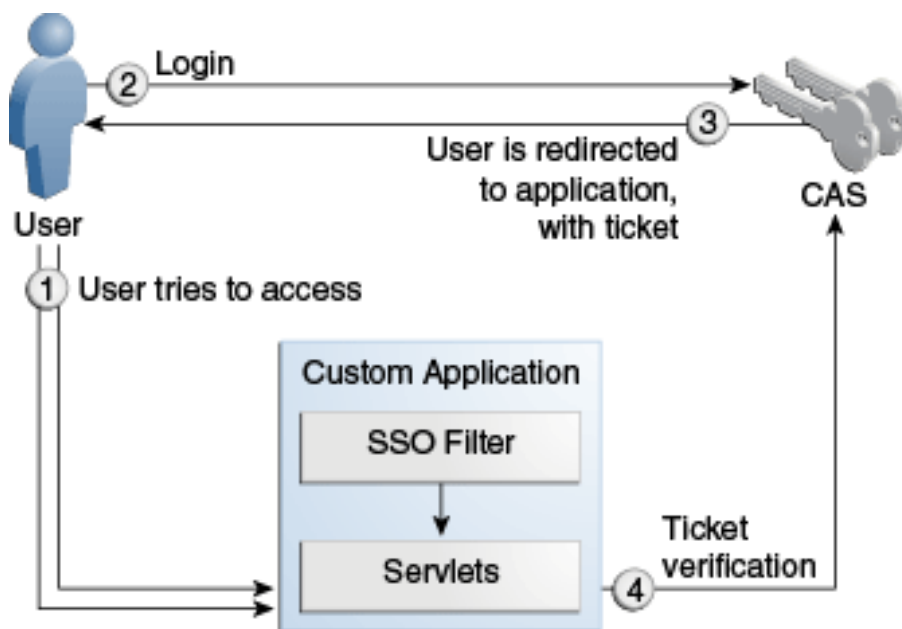
2. **Subsequent Access**

   a. When the user attempts to access another application protected by CAS, the user is redirected to the CAS login page.

   b. The cookie is retrieved from the request, implicit login is performed, and the login page is bypassed.

   c. The user is redirected back to the application with a ticket.

   d. The application verifies the user's identity by verifying the ticket against CAS.

58.5 Authorization Model

General administrator grants users access to applications using WEM Admin. You can simplify the administrator's job by coding a predefined user.

General administrator uses WEM Admin to couple objects. You code a predefined user for the application. How the user fits into the authorization model is explained below.

In the figure below, Site, Application, User, and Role each have a counterpart menu option in WEM Admin. ACLs and groups are exposed on each user's page.

Figure 58-3   Authorization Model
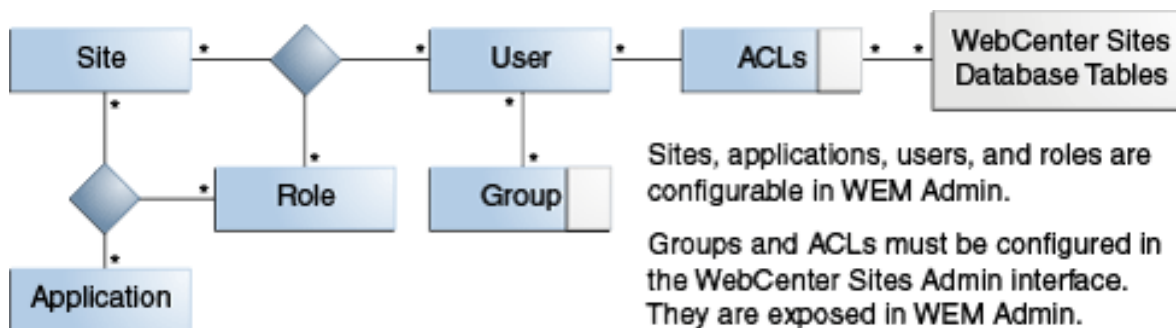
Figure 58-4   WEM Admin Menu Bar
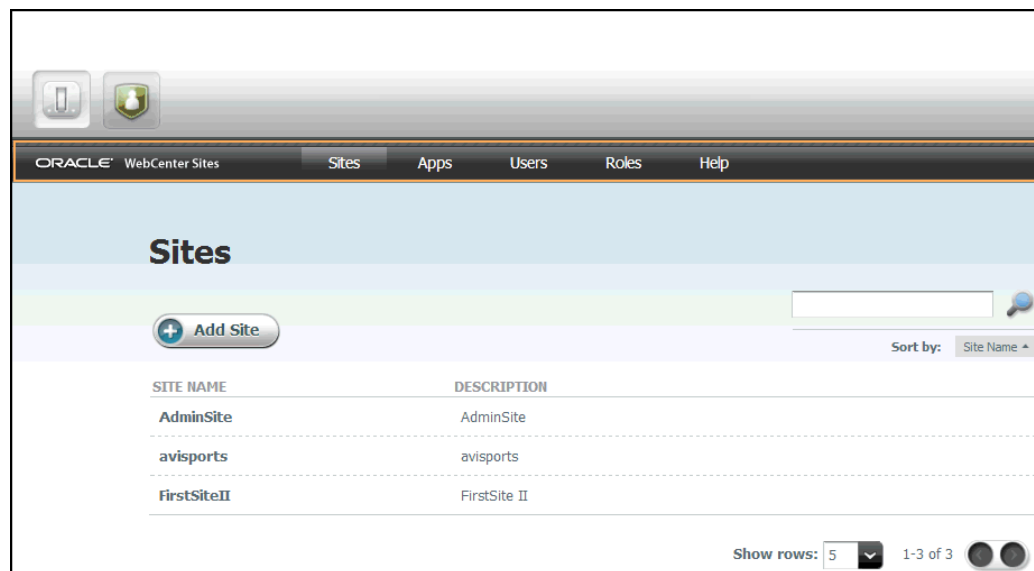
Authorization is managed at three levels: application, REST, and database.

- Application-level authorization requires sharing a role to a user and an application on the same site, which grants the user access to the application on that site. The roles of role-protected interface functions must be shared to the application users.

- REST-level authorization regulates the user's permission to operate on the application's resources, *assuming ACLs are correctly assigned.* REST-level authorization requires configuring groups with privileges to operate on objects

that map to REST resources. Users who are assigned to a group gain the group's privileges.

Developers can define a user in their applications (by user name and password) to act as a proxy for logged-in users, which eliminates the need for administrators to configure REST security for each logged-in user. Once an application is deployed and registered, a general administrator authorizes its predefined user by: 1) configuring the predefined user in WEM Admin for application access, 2) configuring a group (in the Admin interface) with privileges to operate on the applications' resources, and 3) assigning the predefined user to the group (by using either the WEM Admin or the Admin interface). The group's privileges are passed to the predefined user and then to logged-in users when they access the application. Supported security configurations are described and listed in REST Authorization. The Articles sample application provided with the WEM Framework specifies a predefined user.

- At the database level, ACLs determine the individual user's access to the system, that is, permission to log in and operate on the database, *regardless of the user's membership in any groups.* Membership in a group does not grant permissions to a user who lacks the appropriate ACLs and permissions to the database tables.

  Default ACLs give users almost unrestricted permissions, but not the means, to operate on objects in many of the database tables. Those permissions are modulated at the REST level: Either directly by the user's membership in groups (in the absence of a predefined user), or indirectly by the application's predefined user and his membership in groups. Modifying a group's privileges to operate on objects modifies the group member's privileges to operate on resources. The same user on the WebCenter Sites side remains unaffected by group memberships. Permissions to content are still regulated by ACLs and actuated by sites and roles.

58.6 Custom Applications

With WEM you can develop custom applications that you can implement in a loosely coupled manner to the content management platform. The applications you develop will use the REST API Web services and SSO mechanism enabled by the WEM Framework. You deploy these applications to an application server different from the platform's application server. You can, therefore, write custom applications independently of the platform's deployment infrastructure.

Most custom applications are deployed remotely.

Figure 58-5   Remote Application Deployment
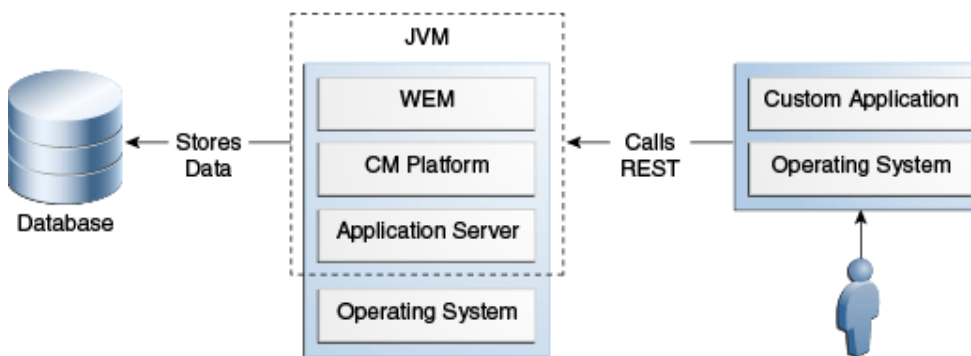
Custom applications can be implemented as content management or delivery applications. We recommend getting started with the content management side, as it typically does not require much performance tuning effort.

The WEM Framework ships with several lightweight sample applications, which you can launch and analyze as models for developing your own applications. Articles illustrates a content management application. Specifications can be found in Developing Applications with WEM Framework, source code is provided in the WebCenter Sites `Misc/Samples` folder, and other supporting information is provided in the REST API resource and Bean references. The SSO sample application is for authentication on live sites and the Recommendations application illustrates the creation of REST resources.

58.7 Requirements for REST Resources

A REST request requires a header with a key and a value of a CAS ticket or session Id.

To authenticate all REST `POST/PUT/DELETE` requests as valid, each request requires a header with the `X-CSRF-Token` as the key and a value of either a CAS ticket (multi or single) or a `sessionid`.

59

Working with the Articles Sample Application

The Articles sample application is a simple content management application for managing article assets. When you work with the application's richly documented source code and a self-installation process, it will help you gain knowledge that you'll find useful in developing applications.

Topics:

- About the Articles Sample Application

- Launching the Articles Sample Application

- Testing the Articles Application

59.1 About the Articles Sample Application

The Articles home page displays two articles that you can edit directly in WEM from the custom interface. In this application, you can use the WebCenter Sites REST API to perform a search query from Java code and a modification asset query from JavaScript code. You will be able to run the Articles application and REST services on different application servers.

Cross-domain restrictions in JavaScript prevent AJAX calls directly from the Articles application to the REST services. Therefore, a simple `ProxyController` is introduced to redirect calls from JavaScript to WEM REST Web Services. Custom implementations may reuse this controller implementation.

The application's home page looks like this figure:

Figure 59-1   Home Page

The Articles application is based on the Spring MVC framework. Articles includes a predefined administrative user named `fwadmin` with password `xceladmin`, who is assigned to the REST group named `RestAdmin`. The application's self-installer contains specifications for registering the Articles application and installing its asset model and sample articles. The application does not have internally configured sites or role-protected functions. It has a single, iframe view. Additional specifications are available in Working with the Articles Sample Application.

59.2 Launching the Articles Sample Application

To be able to launch the Articles sample application, you build and deploy it, and then run the installer.

Topics:

- Building and Deploying the Articles Application

- Registering the Articles Sample Application

59.2.1 Building and Deploying the Articles Application

1. Determine or create the site to which you will assign the sample Articles application. The default site is FirstSiteII (a sample WebCenter Sites CM site). It is possible that FirstSiteII is not installed on your system.

    To select or create a site, log in to WEM Admin at the URL:

    ```
    http://<server>:<port>/<cs_application_context>/login
    ```

using the credentials of a general administrator (`fwadmin/xceladmin` are the default values).

> **Note:**
>
> In step 5, you specify the site you have chosen here, which allows the installer to enable the application's asset model and assets on that site.

2. Download and install Java Development Kit 8 Update 51 (or later).

3. Download the latest Apache Ant from `http://ant.apache.org/` and place the Ant `bin` directory into the system `PATH`.

4. Copy `servlet-api.jar` to the Articles application `lib` folder. The `jar` file can be taken from your application server's home directory (for example, Tomcat's `servlet-api.jar` is located in the `home lib` directory).

5. Set the following parameters in the `applicationContext.xml` file (in `src \articles\src\main\webapp\WEB-INF\`):

   - `casUrl`: Specify the URL of the CAS application: `http://<server>:<port>/<context_path>`

   - `csSiteName`: Specify the name of the site that you selected in step 1.

   - `csUrl`: Specify the URL where the WebCenter Sites platform is running: `http://<server>:<port>/<context>`

   - `csUserName`: The default value is `fwadmin`. This is the application's predefined user, a general administrator with membership in the `RestAdmin` group which has unrestricted permissions to REST services. If you specify a different user, you must name a user equivalent to `fwadmin`. For instructions about creating a general administrator, see Creating Users in *Oracle Fusion Middleware Administering Oracle WebCenter Sites*.

   - `csPassword`: Specify the predefined user's password.

   - `articlesUrl`: Point to the URL where the sample application is accessed.

6. Run the Ant build with the default target (enter `ant` on the command line).

7. Deploy the resulting `target/articles-1.0.war` to an application server.

   On deployment, the following content is copied from source to target: The contents of the `lib` folder are copied to `/WEB-INF/lib`. The contents of the `resources` folder are copied to `/WEB-INF/classes/`. For information about the structure of the source application, see Developing Applications with WEM Framework.

> **Note:**
>
> The mcast port in `cas-cache.xml` is not replaced by the installer. The mcast port is left as `@casCacheMultiCastGroupPort@` and the application will fail to deploy unless manually edited.
>
> A solution is to copy `cas-cache.xml` from WebCenter Sites into the Articles application. This can eliminate some problems associated with manually updating the mcast address and port.

8. In the `customBeans.xml` file (located in the `sitesinstall/config` folder), specify the trusted URLs CAS can redirect a visitor to upon successful login. For example, add the following URL to the `customBeans.xml` file:

   ```
   <value>http://<hostname>:<portnumber>/articles-1.0/*</value>
   ```

59.2.2 Registering the Articles Sample Application

The Articles application has a self-installer, which starts running when you log in to the `install.app` page. The installer registers the sample application (including the view) and creates its data model and assets in the WebCenter Sites database.

> **Note:**
>
> Specifications for the registration asset types `FW_View` and `FW_Application` can be found in the *Oracle Fusion Middleware Java API Reference for Oracle WebCenter Sites* and in Registering Applications Manually in WEM Framework.

To run the Articles installer:

1. Navigate to the `install.app` page:

   ```
   http://<hostname>:<portnumber>/<context_path>/install.app
   ```

   For example:

   ```
   http://localhost:9080/articles-1.0/install.app
   ```

2. Use any credentials to log in.

   The application's predefined user, specified by `csUserName` and `csPassword`, provides you with permissions to the application. The sample application does not perform authorization checks as it does not use roles.

3. The self-installation process invokes `InstallController.java`, which first registers the application (including the view, in an application Bean), then writes the sample asset type and assets to the database.

   a. `InstallController.java` registers the Articles application with the WEM Framework:

      - `InstallController.java` creates an application asset named `Articles` (asset type `FW_Application`) in the WebCenter Sites database.

The `iconurl` attribute points to the URL where the icon representing the application is located.

The `layouturl` attribute specifies the URL of the `layout.app` page (implemented by `LayoutController.java`). The `layout.app` page defines the application layout.

The `layouttype` attribute takes the default (and only) value: `layoutrenderer`. Using the `layoutrenderer` value, the UI container is responsible for rendering the application's associated views by using the `layout.app` page, specified by `layouturl`.

- `InstallController.java` creates a view asset named `ArticlesView` (asset type `FW_View`) in the WebCenter Sites database. The association between the view asset and the application asset is made through the `views` attribute in the `FW_Application` asset type.

b. `InstallController.java` installs the application's asset model and sample assets:

- Creates the application's `FW_Article` asset type in the WebCenter Sites database. (`FW_Article` is a basic asset type defined in `InstallController.java`.)

- Enables the `FW_Article` asset type on the site that was specified in the `csSiteName` parameter in `applicationContext.xml` (step 5 of Building and Deploying the Articles Application).

- Writes the two sample article assets to the `FW_Article` asset type tables. (The articles' text and images are stored in: `/sample app/articles/src/main/resources/install`.)

c. `InstallController.java` creates an asset type-based index to support searches on assets of type `FW_Article`. (The controller specifies index configuration data.)

4. When the installation process completes successfully, `InstallController.java` displays a confirmation (at `http://<server>:<port>/articles/install.app`), that the sample data imported successfully and directing you to the **Home** page (`home.app`).

59.3 Testing the Articles Application

Try out the Articles sample application. All you need to do is navigate to the home page and log in.

To test the Articles application:

1. Navigate to the `home.app` page:

```
http://<hostname>:<portnumber>/<context_path>/home.app
```

For example:

```
http://localhost:8080/articles-1.0/home.app
```

2. Use any credentials to log in.

The application's predefined user, specified by `csUserName` and `csPassword`, provides you with permissions to the application. The sample application does not perform authorization checks as it does not use roles.

WEM displays the application's home page.

3. To experiment with this application (for example assign it to other sites and add users), use WEM Admin.

60

Developing Applications with WEM Framework

You can explore the Articles sample application to understand the basic architecture of an application that makes REST calls.

For information about developing applications with the WEM framework, see these topics:

- About the Articles Sample Application's Structure

- About the Articles Sample Application's Configuration Files

- Making REST Calls

- Constructing URLs to Serve Binary Data

- Accessing Parameters from the WEM Framework

- Registering Applications with Different Views

60.1 About the Articles Sample Application's Structure

In the source directory for the Articles sample application, you will see source files, logger file, installer resources, home page files, and configuration files.

The following figure shows the source structure of the Articles sample application. On deployment, the following directories are copied from source to target: The contents of the `lib` directory are copied to `/WEB-INF/lib/`. The contents of the `resources` directory are copied to `/WEB-INF/classes/`.

Figure 60-1   Articles Sample Application Source Structure
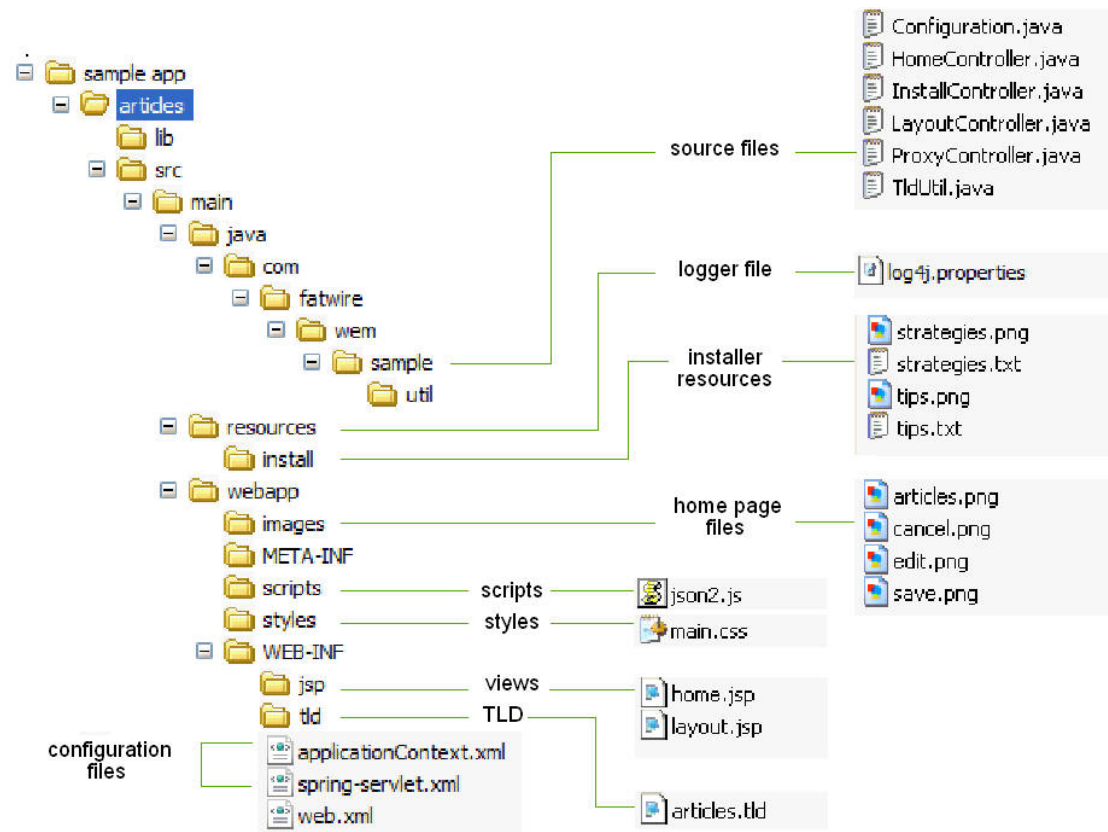
Articles is a Java Web application developed on Spring MVC. The following pages are available:

- /install.app is the Articles installation page, which also displays a confirmation message when the application is successfully installed.

- /home.app is the home page of the Articles application.

60.2 About the Articles Sample Application's Configuration Files

The Articles sample application's configuration files are applicationContext.xml and spring-servlet.xml.

- applicationContext.xml (in /WEB-INF/) holds SSO and application-specific configurations (such as a predefined user and the site on which to enable the data model and assets).

- spring-servlet.xml (in /WEB-INF/) is the default Spring configuration file. This file stores the Spring configuration and references the following controllers (described in Source Files):

  - HomeController

  - InstallController

  - LayoutController

  - ProxyController

– `loggingconfig.xml` (in `/<sitesshared>/config`) is the logging configuration file. On application deployment, it is copied from `/<sitesshared>/config` to `webcentersites\sites-home\template\config`.

Source Files

`/sample app/articles/src/main/java/`

The `/sample/` folder contains the source files listed below:

- `Configuration.java` is populated (by the Spring framework) from the `applicationContext.xml` file.

- `HomeController.java` is the home page controller, which renders a single home page. This controller reads the list of sample articles from the WebCenter Sites platform using the REST API and displays them on the home page.

  The sample articles consist of images and text, stored in `/sample app/articles/src/main/resources/install`. The sample articles are installed in the WebCenter Sites database by `InstallController.java`.

- `InstallController.java` registers the Articles application, and writes the application's asset model and sample assets to the database.

- `LayoutController.java` displays the application's layout page (`layout.app`) used by the WEM UI framework. `LayoutController.java` is also used during the application registration procedure.

- `ProxyController.java` delegates AJAX requests to the WebCenter Sites REST servlet.

- `TldUtil.java` utility class contains TLD function implementations.

Installer Resources

`/sample app/articles/src/main/resources/install`

The `/install/` folder contains the following resources, used by the `InstallController` to construct the home page (Figure 60-3):

- `strategies.png`

- `strategies.txt`

- `tips.png`

- `tips.txt`

Home Page Files

`/sample app/articles/src/main/webapp/images`

The `/images/` folder contains:

- `articles.png` icon (Figure 60-2), which represents the Articles application in the applications bar

- In Figure 60-3:

  – `edit.png` is the icon for the **Edit** function

  – `save.png` is the icon for the **Save** function

- cancel.png is the icon for the **Cancel** function

Scripts

/sample app/articles/src/main/webapp/scripts

The /scripts/ folder contains the json2.js utility script, used to convert strings to and from JSON objects.

Styles

/sample app/articles/src/main/webapp/styles

The /styles/ folder contains main.css, which specifies CSS styles used by this Web application.

Views

/sample app/articles/src/main/WEB-INF/jsp

The /jsp/ folder contains:

- home.jsp, which is used to render the home page view of the Articles application (Figure 60-3).

- layout.jsp, which defines the application layout.

WEB-INF

/sample app/articles/src/main/WEB-INF

The /WEB-INF/ folder contains:

- articles.tld, the TLD declaration file

- spring-servlet.xml, the Spring configuration file

- web.xml, the Web application deployment descriptor

Figure 60-2   Articles Icon (articles.png)

60.3 Making REST Calls

WebCenter Sites REST resources support two types of input and output formats: XML and JSON. If you would like to get specific return formats, use HTTP headers that specify the MIME type `application/xml` or `application/json`.

For example, when specifying input format to be XML, set `Content-Type` to `application/xml`. When specifying the output format, set `Accept` (the expected format) to `application/xml`. If other output formats are specified, they are ignored. The default is XML, if not specified in `Content-Type` or `Accept` (for sample code, see Making REST Calls from JavaScript).

Topics:

- Making REST Calls from JavaScript

- Making REST Calls from Java

60.3.1 Making REST Calls from JavaScript

- Use the following code (in `home.jsp`) to perform AJAX calls to the asset REST services to save asset data. Note that the request is actually performed to the proxy controller which redirects the request to the destination REST service.

> **Note:**
>
> We use the JSON stringify library to serialize a JavaScript object as a string. It is much more convenient to write JSON objects instead of strings.

```
// Form the URL pointing to the asset service
// to the proxy controller, which will redirect this request to the CS REST
servlet.
var idarr = assetId.split(":");
var assetUrl = "${pageContext.request.contextPath}/REST/sites/$
{config.csSiteName}/types/" + idarr[0] + "/assets/" + idarr[1];

// For the data object to be posted.
var data =
{
 "attribute" :
[
{
   "name" : "source",
  "data" :
 {
    "stringValue" : document.getElementById("source_e_" + assetId).value
}
},
{
  "name" : "cat",
 "data" :
 {
"stringValue" : document.getElementById("cat_e_" + assetId).value
}
}
],
 "name" : document.getElementById("name_e_" + assetId).value,
 "description" : document.getElementById("desc_e_" + assetId).value,
 // This should be removed.
 "publist" : "${config.csSiteName}"
};
// Convert JSON data to string.
var strdata = JSON.stringify(data);

// Perform AJAX request.
var req = getXmlHttpObject();
req.onreadystatechange = function ()
{
if (req.readyState == 4)
{
if (req.status == 200)
 {
 // On successful result
// update the view controls with new values and switch the mode to 'view'.
  for (c in controls)
{
document.getElementById(controls[c] + "_v_" + assetId).innerHTML =
document.getElementById(controls[c] + "_e_" + assetId).value;
}
switchMode(assetId, false);
}
else
{
```

```
// Error happened or the session timed out,
// reload the current page to re-acquire the session.
alert("Failed to call " + assetUrl + ", " + req.status + " " + req.statusText);
window.location.reload( false );
}
}
};
// We put Content-Type and Accept headers
// to tell CS REST API which format we are posting
// and which one we are expecting to get.
req.open("POST", assetUrl, true);
req.setRequestHeader("Content-Type", "application/json;charset=utf-8");
req.setRequestHeader("Content-Length", strdata.length);
req.setRequestHeader("Accept", "application/json");
req.send(strdata);
}
```

60.3.2 Making REST Calls from Java

- Use the code below (in `HomeController.java`) to call the assets search service to list all assets of type `FW_Article`. The code uses the Jersey Client library passing objects from the `rest-api-<version>.jar` library provided by the WEM Framework. This leverages strong typing in Java.

  It is important to note that a token must be acquired from Java code by calling the `SSOAssertion.get().createToken()` method. It is unnecessary to do so in JavaScript as that side is authenticated against WEM SSO.

```
// Use Jersey client to query CS assets.
Client client = Client.create();
String url = config.getRestUrl() + "/types/FW_Article/search";
WebResource res = client.resource( url );

// Construct URL and add token (for authentication purposes)
// and fields (specify which fields to retrieve back) parameters.
res = res.queryParam("fields",
URLEncoder.encode("name,description,content,cat,source", "UTF-8"));
res = res.queryParam("ticket",
SSO.getSSOSession().getTicket(res.getURI().toString(), config.getCsUsername(),
config.getCsPassword()));
// Put Pragma: auth-redirect=false to avoid redirects to the CAS login page.
Builder bld = res.header("Pragma", "auth-redirect=false");

// Make a network call.
AssetsBean assets = bld.get(AssetsBean.class);
```

Note:

The custom `Pragma: auth-redirect=false` header instructs the CAS SSO filter not to redirect to the CAS sign-in page, but to return a 403 error instead, when no ticket is supplied or the supplied ticket is invalid.

60.4 Constructing URLs to Serve Binary Data

For the Articles application, you can leverage the Blob server in WebCenter Sites to serve BLOB data. You can use the `getBlobUrl` function to construct a URL pointing to the binary data for a given attribute in a given asset.

- `blobUrl` points to the Blob server (`http://localhost:8080/cs/BlobServer` by default.

```
public String getBlobUrl(String assetType, String assetId, String attrName,
String contentType)
throws Exception
{
String contentTypeEnc = URLEncoder.encode(contentType, "UTF-8");

return blobUrl + "?" +
"blobkey=id" +
"&blobnocache=true" +
"&blobcol=thumbnail" +
"&blobwhere=" + assetId +
"&blobtable=" + assetType +
"&blobheader=" + contentTypeEnc +
"&blobheadername1=content-type" +
"&blobheadervalue1=" + contentTypeEnc;
    }
```

- Alternatively, to get binary data, load an asset using the resource `/sites/{sitename}/types/{assettype}/assets/{id}`. When loaded, the asset contains the URL pointing to the BLOB server.

60.5 Accessing Parameters from the WEM Framework

With the `WemContext` JavaScript Context object, you can empower applications to find out which user has logged in to which site. You can also enable applications to share data.

The UI container provides a JavaScript Context object (`WemContext`) to all applications inside the container. The Context object is used by the applications to get details from the WEM Framework about the logged-in user and site (typically, to get the current site's name from the UI container). The Context object also provides various utility methods that the applications use to share data. The Context Object can be used by applications running in the same domain as WebCenter Sites or in different domains.

> **Note:**
>
> The `wemcontext.html` file lists the exposed methods, summarized in Methods Available in Context Object.

Topics:

- Initializing and Using Context Object in the Same Domain
- Initializing and Using Context Object for Cross-Domain Applications
- Methods Available in Context Object

60.5.1 Initializing and Using Context Object in the Same Domain

To initialize and use Context Object for applications in the WebCenter Sites domain:

1. Include `wemcontext.js`.

2. Retrieve an instance of the `WemContext` object.

3. Use the methods of `WemContext`.

```
<script src='http://<csinstalldomain>/<contextpath>/wemresources/js/WemContext.js'></
script>
<script type="text/javascript">
var wemContext = WemContext.getInstance(); // Instantiate Context Object
var siteName = wemContext.getSiteName(); // Get Site Name
var userName = wemContext.getUserName(); // Get UserName
</script>
```

60.5.2 Initializing and Using Context Object for Cross-Domain Applications

To initialize and use Context Object for cross-domain applications:

1. Copy `wemxdm.js`, `json2.js`, and `hash.html` (from the `Misc/Samples` folder) to your application.

2. Open the `sample.html` file and make the following changes to perform cross-domain calls:

- Change the paths of `wemxdm.js` and `json.js` and `hash.html` to their paths in the application (see the example after step d).

- Change the path of `wemcontext.html` to its location in WebCenter Sites . (`wemcontext.html` is located under `/wemresources/wemcontext.html`. Use the WebCenter Sites host name and context path.)

- In the interface declaration, specify methods to be used in the framework.

- Implement those methods in the local scope and invoke the remote method.

```
<script type="text/javascript" src="../js/wemxdm.js"></script>
<script type="text/javascript">
//  Request the use of the JSON object
WemXDM.ImportJSON("../js/json2.js");
var remote;
window.onload = function() {
// When the window is finished loading start setting up the interface
remote = WemXDM.Interface(/** The channel configuration */
{
// Register the url to hash.html.
local: "../hash.html",
// Register the url to the remote interface
remote: "http://localhost:8080/cs/wemresources/wemcontext.html"
}, /** The interface configuration */
{
remote: {
getSiteName :{},
...

}
},/**The onReady handler*/ function(){
// This function will be loaded as soon as the page is loaded
populateAttributes();
});
}
</script>

<script type="text/javascript">
/** Define local methods for accessing remote methods  */
function getSiteName(){
```

```
remote.getSiteName(function(result){
alert("result = " + result);
});
}
 ...
</script>
```

60.5.3 Methods Available in Context Object

The following table lists and describes the methods available in Context Object.

Table 60-1   Methods Available in Context Object

Return Type	Method name and Description
Object	getAttribute(attributename) Returns attribute value for the given attribute name.
Object	getAttributeNames() Returns all the attribute names.
Object	getCookie(name) Returns cookie value for the given name. It has all restrictions of the normal browser cookie.
Object	getCookies() Returns all the cookies.
Object	getLocale() Returns locale.
Object	getSiteId() Returns the site ID.
Object	getSiteName() Returns the site name.
Object	getUser() Returns user object.
Object	getUserName() Returns user name.
void	removeCookie(name, properties) Removes cookie.
void	setAttribute(attributename, attributevalue) Sets attribute. These attributes can be accessed in other applications.
void	setCookie(name,value,expiredays,properties) Sets the cookie.

60.6 Registering Applications with Different Views

In WEM Framework, when you register an application to expose it, an asset of type `FW_Application` and another of type `FW_View` are created for each view associated with the application. These asset types are enabled on AdminSite.

Their attributes are defined in the *Oracle Fusion Middleware Java API Reference for Oracle WebCenter Sites*. Programmatic registration is the preferred method. For an example of manual registration, see Registering Applications Manually in WEM Framework.

Topics:

- Registering Applications with an iframe View

- Registering Applications with JavaScript and HTML Views

60.6.1 Registering Applications with an iframe View

The section uses code from the Articles sample application to illustrate the registration process. Articles has a single view of type iframe. The same steps apply to JavaScript and HTML views.

To register an application:

1. Create or get an icon to represent your application. (The icon is displayed in the applications bar.)

   (The Articles sample application uses the `articles.png` image file located in: `/sample app/articles/src/main/webapp/images/`)

2. Create a file that specifies the layout of the application in HTML, that is, for each view, create a placeholder element to hold the content rendered by the view. Applications and views are related as shown in the next figure.

   For example, `layout.jsp` for the Articles sample application contains the following line:

   ```
   <div id="articles" style="float:left;height:100%;width:100%"
   class="wemholder"></div>
   ```

   The view's content is rendered within the placeholder element when the application is displayed (`layout.app` renders the application's layout; `home.app` renders the view).

   > **Note:**
   >
   > When creating the layout file, specify a unique `id` for the placeholder element. Specify the same `id` for the `parentnode` attribute when coding the `view` object. Use `class="wemholder"` for the placeholder elements.

Figure 60-4 Applications and Views
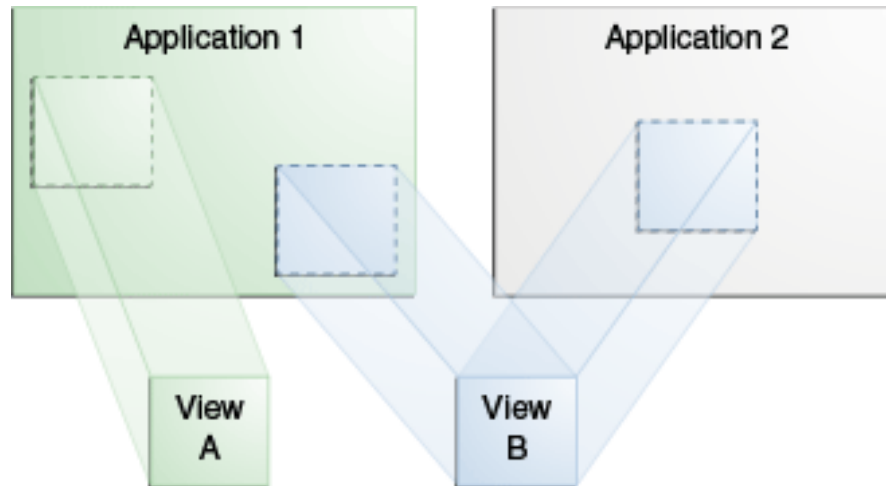
The relationship between applications and views is many-to-many. One application can have multiple views and each view can be used by many applications. Only registered views can be shared (through their asset IDs). The view is created within the context of its application if the asset ID is omitted. In the basic case, an application has only one view associated with it.

3. Invoke the `PUT wem/applications/{applicationid}` REST service and specify your application bean.

4. Populate the bean with the view asset and application asset.

   For an iframe view, use the code of the Articles sample application, that is, `InstallController.java` (locate the comment lines `// Create a new view object` and `// Create a new application object`). Set the `layouturl` attribute to specify the URL of the application's layout page.

   In the Articles application, the `layouturl` attribute points to the URL of `layout.app` (implemented by `LayoutController.java`):

   ```
   app.setLayouturl(config.getArticlesUrl() + "/layout.app");
   ```

5. To test the results of your registration process, log in to the WEM Admin interface as a general administrator and select **Apps** on the menu bar. Your application should be listed on that page.

60.6.2 Registering Applications with JavaScript and HTML Views

For applications that use HTML and JavaScript views, follow the steps in Registering Applications with an iframe View, but use the sample code and attributes listed in the following sections:

- Rendering JavaScript View

- Rendering HTML View

60.6.2.1 Rendering JavaScript View

Note:

JavaScript specified in the view is rendered (executed) when the application is rendered. Ensure the JavaScript does not conflict with other views.

Sample code:

```
window.onload = function () {
   if (GBrowserIsCompatible()) {
     var map = new GMap2(document.getElementById("map_canvas"));
     map.setCenter(new GLatLng(37.4419, -122.1419), 13);
     map.setUIToDefault();
   }
}
```

- **Rendering the JavaScript view from a source URL**

  Set the following attributes:

  - `name`: Name of the view

  - `parentnode`: ID of the placeholder element (from step 2 in Registering Applications with an iframe View)

  - `viewtype`: `fw.wem.framework.ScriptRenderer`, which renders JavaScript into the placeholder element

  - `sourceurl`: Path of the `.js` file, which provides content for the view. For example: `http://example.com:8080/js/drawTree.js`

- **Rendering the JavaScript view from source code**

  Set the following attributes:

  - `name`: Name of the view

  - `parentnode`: ID of the placeholder element (from step 2 in Registering Applications with an iframe View)

  - `viewtype`: `fw.wem.framework.ScriptRenderer`, which renders JavaScript into the placeholder element

  - `javascriptcontent`: JavaScript code (sample provided above). The code must not contain `<script>` tags.

60.6.2.2 Rendering HTML View

Note:

HTML specified in the view is rendered (executed) when the application is rendered.

Sample code:

```
<object width="480" height="385">
 <param name="movie" value="http://www.localhost:8080/jspx/flash_slider_main.swf"></
param>
 <param name="allowFullScreen" value="true"></param>
 <embed src=" http://www.localhost:8080/jspx/flash_slider_main.swf"
  type="application/x-shockwave-flash" allowscriptaccess="always"
allowfullscreen="true"
  width="480" height="385">
 </embed>
</object>
```

- **Rendering the HTML view from a source URL**

  Set the following attributes:

  - `name`: Name of the view

  - `parentnode`: ID of the placeholder element (from step 2 in Registering Applications with an iframe View)

  - `viewtype`: `fw.wem.framework.IncludeRenderer`, which renders JavaScript into the placeholder element

  - `sourceurl`: Path to the HTML file that provides content for the view. For example: `http://example.com:8080/js/drawTree.jsp`.

- **Rendering the HTML view from source code**

  Set the following attributes:

  - `view`: Name of the view

  - `parentnode`: ID of the placeholder element (from step 2 in Registering Applications with an iframe View)

  - `viewtype`: `fw.wem.framework.IncludeRenderer`, which renders JavaScript into the placeholder element

  - `includecontent`: HTML content (sample provided above. The code must not contain `<html>` or `<body>` tags.

61

Developing Custom REST Resources with WEM Framework

Within the WEM Framework, you can develop custom REST resources as the Recommendation sample application demonstrate.

Topics:

- Creating REST Resources for WebCenter Sites and Satellite Server: Example
- Creating REST Resources

61.1 Creating REST Resources for WebCenter Sites and Satellite Server: Example

Through the Recommendation sample application you can learn how you can create REST resources for WebCenter Sites and Satellite Server.

The application registers a new REST resource `sample/recommendations/<id>` with `GET` and `POST` operations, which allow for retrieval and modification of static list recommendations. The application also demonstrates how it is possible to leverage the Satellite Server caching system.

Topics:

- Building and Deploying the Recommendations Sample Application
- Testing the Recommendations Sample Application

61.1.1 Building and Deploying the Recommendations Sample Application

1. The Recommendations sample application is located in the `Misc/Samples` folder under your WebCenter Sites installation directory. Navigate to `recommendations` and edit the `build.properties` file. Specify the correct paths for `cs.webapp.dir` and `ss.webapp.dir` properties.

2. Run Apache `ant` while in the `recommendations` folder.

   This will build and deploy your sample application.

3. Launch the `catalogmover` application. Use the **Server, Connect** menu to connect to WebCenter Sites. Go to **Catalog, then Auto Import Catalog(s)** and select `src\main\schema\elements.zip` file. Append `xceladmin, xceleditor` when specifying the list of ACLs.

4. Go to the WebCenter Sites web application folder. Edit the `WEB-INF/classes/custom/RestResource.xml` file. Uncomment `recommendationService`, `recommendationConfig` and `resourceConfigs` beans.

5. Go to the Satellite Server web application folder. Edit `WEB-INF/classes/custom/RestResource.xml` file. Uncomment `recommendationService`, `recommendationConfig`, and `resourceConfigs` beans.

6. Restart both WebCenter Sites and Satellite Server.

61.1.2 Testing the Recommendations Sample Application

You can test the Recommendations sample application as follows:

- Use the existing static list recommendation ID (or create a new recommendation) for the URL `http://<hostname>:<port>/<contextpath>/REST/sample/recommendations/<recommendationid>`.

- Use the same URL for both WebCenter Sites and Satellite Server installations. For example, use `http://localhost:8080/cs/REST/sample/recommendations/1266874492697`. See the XML response for both WebCenter Sites and Satellite Server.

61.2 Creating REST Resources

This section includes the following topics:

- About the Recommendations Sample Application's Structure

- Implementing Custom REST Resources

61.2.1 About the Recommendations Sample Application's Structure

The Recommendations sample application was created to guide you through the process of creating your own REST resources.

Figure 61-1   Recommendations Sample Application
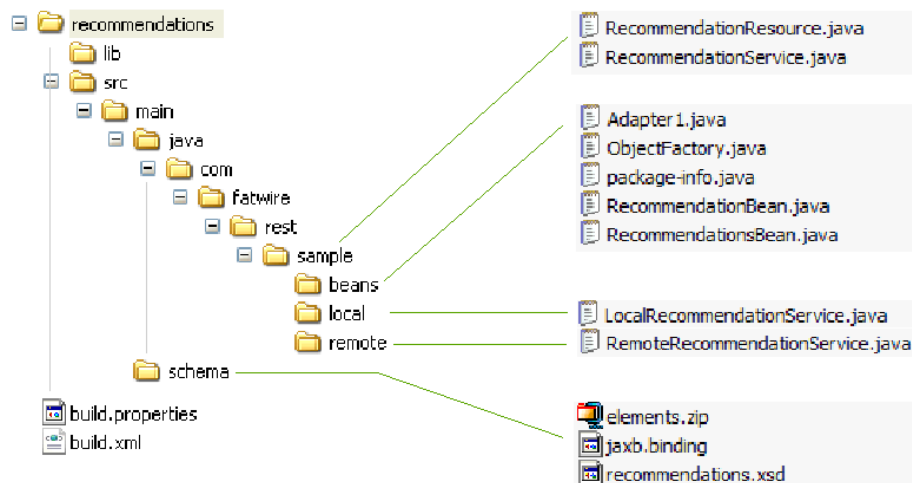
- Schema files: `src/main/schema`

  - `elements.zip` contains a sample element, which is used by Satellite Server for caching purposes.

  - `jaxb.binding` is a customization for the default JAXB bindings used during the bean generation process.

    –   `recommendation.xsd` is an XML schema for the `RecommendationService` beans.

- Java source files: `src/main/java/ ... /sample`

    –   `RecommendationResource` contains the REST resource implementation. It is used on both WebCenter Sites and Satellite Server.

    –   `RecommendationService` is an interface that provides the functionality for the `RecommendationResource` class. It is implemented differently, depending on where the resource is hosted: locally (on WebCenter Sites) or remotely (on Satellite Server).

    –   `beans/*` classes are generated using Java `xjc` compiler. They are pre-packaged with the application. To regenerate beans (that is, when changing the `recommendation.xsd` file), run `generate` Ant's task from `build.xml`.

    –   `LocalRecommendationService` is a local (WebCenter Sites) implementation for the `RecommendationService` interface.

    –   `RemoteRecommendationService` is a remote (Satellite Server) implementation for the `RecommendationService` interface.

61.2.2 Implementing Custom REST Resources

1. Write your XSD file describing your REST service (`recommendations.xsd` file).

2. Generate beans using the `JAXB xjc` utility (`generate` Ant's task).

3. Create your REST interface, which will be implemented differently for WebCenter Sites and Satellite Server.

4. Implement the REST interface by extending the following classes:
   `com.fatwire.rest.BaseLocalService`
   `com.fatwire.rest.BaseRemoteService`

5. This step is optional in case you decide to leverage Satellite Server caching:

   Create elements on the WebCenter Sites side, which load the same assets as the local implementation does.

6. Create your REST resource class by extending the `com.fatwire.rest.BaseResource` class.

7. Register your REST service and configuration in `WEB-INF/classes/custom/RestResources.xml` file on both WebCenter Sites and Satellite Server sides.

   The `custom/RestResources.xml` file contains the following components:

   - The only mandatory bean is the bean with `resourceConfigs` ID. The `resourceConfigs` property contains references to all REST configurations used.

   > **Note:**
   >
   > If custom `resourceConfigs` is uncommented, then bean should be referenced. Otherwise, the default REST resource, which is provided with the WEM installation is not registered.

- Resource configurations must be of type
  `com.fatwire.rest.ResourceConfig`. Typically only one instance of this
  class is registered (multiple services can be registered per configuration).

  > **Note:**
  >
  > For multiple services, create a new configuration for each disjoint group of
  > your REST services, usually identified by separate XSD files.

- The `resourceClasses` property contains the list of all resources used.

- `beanPackage` contains the Java package name specified for the output beans
  when running the `xjc` utility.

- `schemaLocation` is the `xsi:schemaLocation` attribute to be put in all
  output XML files produced by your REST service.

62

Working with Single Sign-On for Production Sites

Here is a simple example that helps you understand how you can enable single sign-on and sign-out for applications on live sites where you won't be able to secure applications with ready-to-use CAS.

Topics:

- Deploying the SSO Sample Application

- Understanding SSO Sample Application's Structure

- Implementing Single Sign-On

- Implementing Single Sign-Out

62.1 Deploying the SSO Sample Application

All you need to do is unpack the WAR file, modify the context file, and deploy the application.

1. Unpack the `wem-sso-api-cas-sample.war` file (to the `/sso-sample` folder, for example). The application is located in the WebCenter Sites installation directory in the `Misc/Samples/WEM Samples/WEM Sample applications/` directory.

2. Modify the `applicationContext.xml` file in the `WEB-INF` folder by setting the following properties:

   - `casUrl`: Point to the CAS server base path:

     ```
     http://localhost:8080/cas
     ```

   - `casLoginPath`: Include the login form template hosted by the SSO sample application:

     ```
     /login?wemLoginTemplate=http%3A%2F%2Flocalhost%3A9080%2Fsso-cas-sample
     %2Ftemplate.html
     ```

3. Deploy the modified SSO sample application to your application server.

4. Access the application.

The SSO sample application consists of the following pages:

- **Protected area**: A page that is protected by the WEM SSO filter. This page contains two single sign-out links.

Figure 62-1   Protected Page with Single Sign-Out Links

You are in the protected area

Hello, fwadmin

Assertion attributes
timeout:900
currentACL:
ElementEditor,RemoteClient,VisitorAdmin,TableEditor,xcelpublish,PageEditor,UserReader,xceleditor,xceladmin,WSEditor,Browser,Visitor,WSAdmin,WSUse:
username: fwadmin
distinguishedName:fwadmin
displayName:fwadmin
Sign out with redirect

Sign out without redirect

The first link (single sign-out with redirect) is an HTML link that performs single sign-out on the CAS side and redirects the user back to the home page. The second link (single sign-out without redirect) is also an HTML link that performs single sign-out on the CAS side, but without leaving or reloading the current page.

- **Public area**: A page that is excluded from the protection filter.

- **Public area with login form**: This page is excluded from the protection filter, but has a login form, which allows performing a sign-in operation without leaving or reloading the current page.

Figure 62-2   Public Area with the Sign in Link

Public area with login form

Sign in

Note:

On successful login CAS redirects the user to the requested service. For security purposes, you may validate the requested service before redirecting the user. Do this by specifying the list of trusted and permitted urls using the Property Management Tool under the System Tools node in the Admin interface. Add the trusted urls (comma separated) to the `valid.urls` property. This property specifies that the trusted URLs users are permitted to use to access WebCenter Sites. Default value of this property is `<wcsites.app.protocol>://<wcsites.app.host>:<wcsites.app.port>/<wcsites.app.contextroot>/*`. See *Oracle Fusion Middleware Property Files Reference for Oracle WebCenter Sites*.

The URLs can be in either of the following formats:

- Exact URL. For instance:

  `http://hostname:port/cs/wem/fatwire/wem/Welcome`

- Matching URL. For instance:

  `http://hostname:port/SitesWebapp/*`

  Where `/*` at the end indicates any URL that has the same prefix as specified

Trusted resources using IP must have URLs specified as well.

62.2 Understanding SSO Sample Application's Structure

With the SSO sample application's basic code for configuring single sign-on and sign-out you can protect applications on production sites.

The following components provide access to the SSO sample application:

- `index.jsp`: Starting page. This page contains links to the pages described as **Protected area**, **Public area**, and **Public area with login form pages** (see Deploying the SSO Sample Application).

- `template.html`: Used to provide a custom sign-in form for CAS. Its path is referenced in the `wemLoginTemplate` parameter in `casLoginPath` in the `applicationContext.xml` file.

Figure 62-3 sso-sample
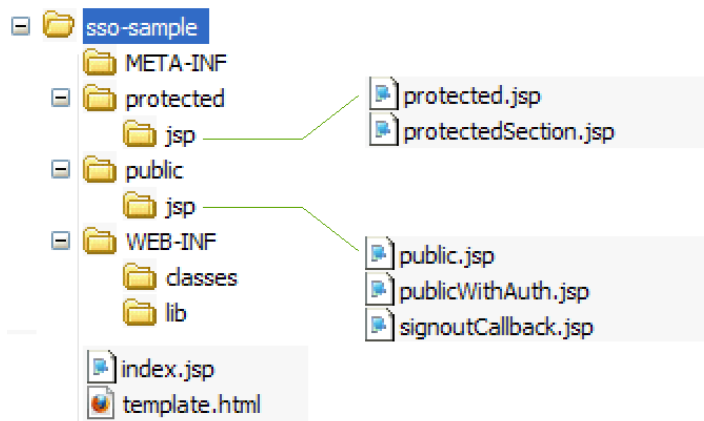
Configuration Files in /sso-sample/WEB-INF

`WEB-INF` contains the following configuration files:

- `applicationContext.xml`: Spring web application configuration file, which configures the SSO subsystem.

- `web.xml`: Web application deployment descriptor.

Protected Files in /sso-sample/protected/jsp

Files in this area are protected by the SSO filter. By default, the following files are included in this folder:

- `protected.jsp`: A page protected by the SSO filter. This page hosts two links for performing single sign-out. The first link leads to the CAS sign-out page with a redirect to the application's home page when sign-out is complete. The second link embeds an iframe into this page, which calls the CAS sign-out page with a redirect to the `signoutCallback.jsp` page. The `protected.jsp` page also prints out all attributes from the `Assertion` object, which describes the current logged in user.

- `protected/jsp/protectedSection.jsp`: Page that is referenced from the `public.jsp` page, when the **Sign in** link is clicked in an embedded iframe. As this page is protected, a login screen is presented in the embedded iframe.

Public Files in /sso-sample/public/jsp

Files in this area are not protected by the SSO filter. By default, the following sample files are included in the `/public/jsp/` folder:

- `public.jsp`: This page not protected by the CAS filter.

- `publicWithAuth.jsp`: This page displays the **Sign in** link. Clicking the link embeds an iframe into the `publicWithAuth.jsp` with the iframe pointing to the `protectedSection.jsp` page. As the page is protected, a login screen is presented in the embedded iframe.

- `signoutCallback.jsp`: This page is called from the `protected.jsp` page upon sign-out completion when using iframe.

62.3 Implementing Single Sign-On

You can implement single sign-on for a website by either presenting the sign-in form when the visitor tries to access a protected page or embedding it into a public page.

- The sign-in form is presented when the visitor tries to access a protected page. This is the default sign-in implementation. This sign in form could be either a default sign-in form shipped with CAS or a custom form provided by an application.

Figure 62-4   Sign-On Form

- The sign-in form is embedded into a public page, and the sign-in function is performed without the user leaving the current page. This behavior can be implemented by embedding the iframe that points to a protected page. As the page is being protected, the sign-in form is presented to the visitor.

Figure 62-5   Sign In Form

62.4 Implementing Single Sign-Out

You implement single sign-out either by retrieving the single sign-out URL or by using an iframe-embedding technique.

- Retrieve the `single sign-out` URL by invoking the following method:

  `getSignoutUrl()` or `getSignoutUrl(String callbackUrl)` method of `com.fatwire.wem.sso.SSO.getSSOSession()` object.

  After performing single sign-out, CAS can optionally redirect to the visitor-supplied URL, which is set in the `callbackUrl` parameter.

- Use an iframe-embedding technique if the sign-out is to be performed without leaving the current page. This technique involves embedding an iframe with the single sign-out URL as source. When the iframe is loaded, the sign-out URL is called (this is done primarily to avoid cross-domain restrictions in browsers).

63

Using REST Resources with the WEM Framework

You can manage assets using the WebCenter Sites REST API if you grant privileges on applications' resources to perform REST operations.

Topics:

- Authentication for REST Resources

- About Configuring CAS

- REST Authorization

- Management of Assets Over REST

63.1 Authentication for REST Resources

The WEM Framework uses the SSO mechanism built on top of CAS for authentication purposes. The system behaves differently when you use the REST API from a browser or programmatically.

When accessing the REST API from a browser, the user is redirected to the CAS login page and, upon successful login, back to the original location with the `ticket` parameter, which is validated to establish the user's identity. When accessing the REST API programmatically, the developer must supply either the `ticket` or `multiticket` parameter.

Both the `ticket` and `multiticket` parameters could be acquired by using either the Oracle SSO API if making calls from Java, or simply by using the HTTP protocol if making calls from any other language. The difference between `ticket` and `multiticket` is that a ticket is acquired per each REST resource and can be used only once (as the name implies, think of a train or a theater ticket, which is valid for one ride or one play), while a multiticket could be used multiple times for any resource. Both the `ticket` and `multiticket` parameters are limited in time, but the typical usage pattern differs. As a ticket is acquired per each call, there is no expiration time. However, reusing the same multiticket will eventually lead to its expiration and getting an `HTTP 403` error. The application must be able to recognize such behavior and fall back to the multiticket re-acquisition procedure in such a case. The decision to use either `ticket` or `multiticket` is up to the application developer.

Topics:

- Acquiring Tickets from Java Code

- Acquiring Tickets from Other Programming Languages (Over HTTP)

- Using Tickets and Multitickets

- SSO Configuration for Standalone Applications

63.1.1 Acquiring Tickets from Java Code

The Oracle SSO API is implemented in an authentication provider-independent manner. Users are not able to register their own SSO authentication providers. Support for a new authentication provider can be implemented only by Oracle. Switching between providers involves only changing the SSO configuration files.

All SSO calls originate at the SSO front-end class SSO. It is used to get the `SSOSession` object. `SSOSession` is acquired per each SSO configuration. It is a single configuration in the web application case, which is loaded using the Spring Web application loader or a configuration loaded from a configuration file in the case of a standalone application.

- To acquire a ticket in a Web application:

```
SSO.getSession().getTicket(String service, String username, String password)
SSO.getSession().getMultiTicket(String username, String password)
```

- To acquire a ticket in a standalone application:

```
SSO.getSession(String configName).getTicket
  (String service, String username, String password)
SSO.getSession(String configName).getMultiTicket
  (String username, String password)
```

63.1.2 Acquiring Tickets from Other Programming Languages (Over HTTP)

The CAS REST API is used to acquire a ticket, or a multiticket, or both in the delivery environment. Two `HTTP POST` calls should be performed to acquire either ticket or multiticket. The difference between ticket and multiticket is that the `service` parameter is * (asterisk) for multiticket, while it is an actual REST resource you are trying to access for the `ticket` parameter.

The example below demonstrates the calls to be made to the CAS server to get a ticket to the `http://localhost:8080/cs/REST/sites` service with `fwadmin/xceladmin` credentials:

1. Call to get `Ticket Granting Ticket`

   Request

```
POST /cas/v1/tickets HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Content-Length: 35

username=fwadmin&password=xceladmin
```

   Response

```
HTTP/1.1 201 Created
Location: http://localhost:8080/cas/v1/tickets/TGT-1-
ej2biTUFoCNBwA5X4lJn4PjYLRcLtLYg2QhLHclInfQqUk3au0-cas
Content-Length: 441
...
```

2. Call to get a Service ticket

   Request

```
POST /cas/v1/tickets/TGT-1-ej2biTUFoCNBwA5X4lJn4PjYLRcLtLYg2QhLHclInfQqUk3au0-
cas  HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Content-Length: 57

service=http%3A%2F%2Flocalhost%3A8080%2Fcs%2FREST%2Fsites
```

Response

```
HTTP/1.1 200 OK
Content-Type: text/plain
Content-Length: 29

ST-1-7xsHEMYR9ZmKdyNuBz6W-cas
```

The protocol is fairly straightforward. First a call to get Ticket Granting Ticket (TGT) is made by passing the user name and password parameter in `application/x-www-form-urlencoded` `POST` request. The Response will contain the `Location` HTTP header, which should be used to issue a second `application/x-www-form-urlencoded` `POST` request with `service` parameter. The response body will contain the actual ticket.

63.1.3 Using Tickets and Multitickets

To use the generated ticket/multiticket, supply the `ticket/multiticket` URL query parameter. For example:

```
http://localhost:8080/cs/REST/sites?ticket=ST-1-7xsHEMYR9ZmKdyNuBz6W-cas
http://localhost:8080/cs/REST/sites?multiticket=ST-2-Bhen7VnZBERxXcepJZaV-cas
```

Figure 63-1   Tickets and Multitickets
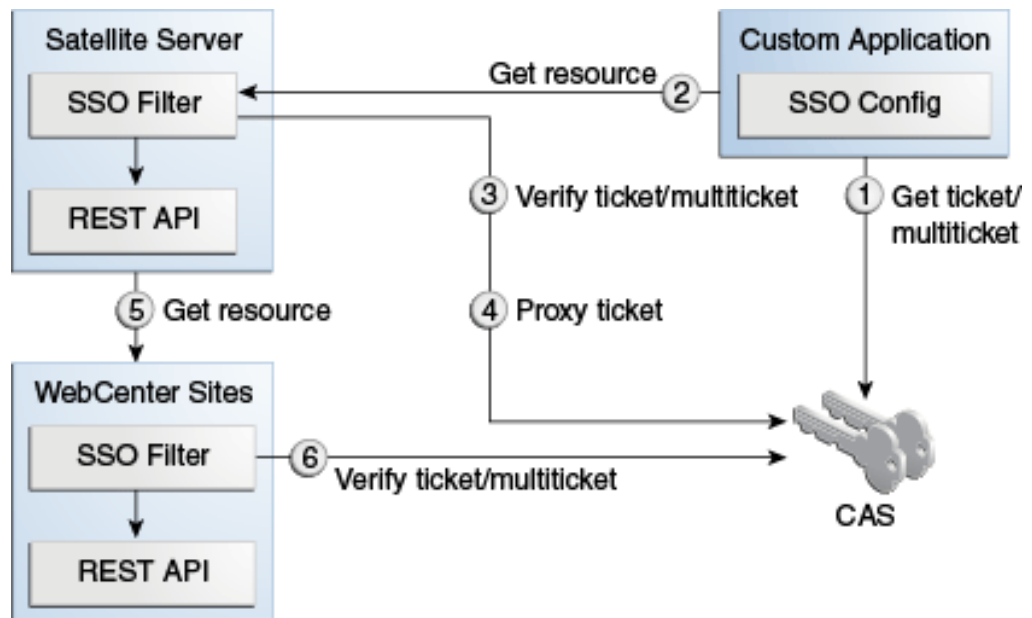
1. The application performs a call to get the ticket/multiticket.

    • Input: service, user name, password

    • Output: ticket /multiticket

2. The application performs call to Remote Satellite Server to get the resource.

   - Input: ticket, resource input data

   - Output: resource output data

3. Remote Satellite Server performs a call to validate the resulting assertion. The assertion contains user information. Satellite Server also maintains a time-based cache of multitickets, so that subsequent calls do not incur the cost of validation.

   - Input: ticket/multiticket

   - Output: assertion

4. This step is optional. The `proxyTickets` parameter, when set to `true` in the `SSOConfig.xml` file on the Satellite Server side, also proxies the ticket.

   - Input: ticket

   - Output: proxied ticket

5. Remote Satellite Server performs a call to WebCenter Sites.

   - Input: assertion (in serialized form), resource input data

   - Output: resource output data

6. This step is optional. If security is enabled on the WebCenter Sites side, it performs a call to validate the ticket.

   - Input: ticket/multiticket

   - Output: assertion

By default the communication channel between WebCenter Sites and Remote Satellite Server is not trusted. The `proxyTickets` parameter in the `SSOConfig.xml` file on Remote Satellite Server is set to `true`, which forces Remote Satellite Server to proxy the ticket supplied by the application that is being accessed.

For optimal performance, the system can be configured for authentication by Satellite Server alone. The security check should be disabled on the WebCenter Sites side by excluding the REST and WebCenter Sites elements used by the REST API from the SSO filter. The `proxyTickets` parameter in the `SSOConfig.xml` file on Remote Satellite Server should be set to `false`. In this mode it is possible to leverage multitickets. Note that the WebCenter Sites installation should be hosted inside a private network in this mode, and the communication channel between WebCenter Sites and Remote Satellite Server should be trusted.

63.1.4 SSO Configuration for Standalone Applications

The single sign-on module relies on the Spring configuration. The only required bean is `ssoprovider`, which references the `ssoconfig` bean.

This section includes the following topics:

- Beans and Properties

- Query Parameters Processed by SSO Filter

63.1.4.1 Beans and Properties

The following table describes `ssolistener` beans and properties.

```
id="ssolistener", class="com.fatwire.wem.sso.cas.listener.CASListener"
```

Table 63-1   id="ssolistener"

Property	Description
No properties for this bean.	n/a

The following table describes `ssofilter` beans and properties.

```
id="ssofilter",
class="com.fatwire.wem.sso.cas.filter.CASFilter"
```

Table 63-2   id="ssofilter"

Property	Description
config	Required. SSO configuration reference. **Sample Value**: ssoconfig
provider	Required. SSO provider reference. **Sample Value**: ssoprovider

The following table describes `provider` beans and properties.

```
id="provider",
class="com.fatwire.wem.sso.cas.CASProvider"
```

Table 63-3   id="provider"

Property	Description
config	SSO configuration reference. **Sample Value**: ssoconfig

The following table describes `config` beans and properties.

```
id="config",
class="com.fatwire.wem.sso.cas.conf.CASConfig"
```

Table 63-4   id="config"

Property	Description
applicationProxyCallbackPath	Proxy callback path, relative to `casUrl`. **Sample Value:** /proxycallback

Table 63-4   (Cont.) id="config"

Property	Description
authRedirect	Use this property to specify the default behavior on unauthenticated access to protected pages. `true` redirects the user to the CAS login page; `false` displays a 403 error if users are not unauthenticated. This setting could be overridden by the Pragma: auth-redirect HTTP header. **Sample Value:** `true`
casLoginPath	Login page path, relative to `casUrl`. Can accept additional query parameters: • `wemLoginTemplate`, points to the page containing the HTML login template to be used instead of the default template. The template must have two input fields: `username` and `password`. Note, that the HTML `<form>` tag should not be used in the template. • `wemLoginCss`, points to the CSS page containing style declarations used on the login form. **Sample Value:** `/login`
casRESTPath	CAS REST servlet path, relative to `casUrl`. **Sample Value:** `/v1`
casSignoutPath	Logout page path, relative to `casUrl`. **Sample Value:** `/logout`
casUrl	Required property. CAS URL prefix. The URL must resolve both internally and externally. **Example**: `http://localhost:8080/cas`
gateway	If `true`, the request to protected pages will be redirected to CAS. If a ticket-granting cookie is present, then the user will be implicitly authenticated; if not, the user will be redirected back to the original location. This is used primarily to allow implicit authentication if the user is logged in to another application.
gateway *(continued)*	Be careful when enabling the redirect behavior to occur by default. Ensure the clients are able to follow the redirects. Otherwise, `gateway=false` URL query parameter should be used to override the default behavior. For example, while processing `wemLoginTemplate` and `wemLoginCss` parameters, CAS does not follow redirects; you will have to prepend `gateway=false` to URLs when turning this setting on. **Default value:** `false`

Table 63-4   (Cont.) id="config"

Property	Description
`multiticketTimeout`	Multiticket timeout in msecs. **Default value:** `600000` The CAS multiticket timeout is based on `cs.timeout`. The timeout expiration calculation starts when the ticket is first used against Web Center Sites.
`protectedMappingExcludes`	List of mappings that should be excluded. Regular expressions are allowed. **Allowed value:** `See protectedMappingIncludes`

`protectedMappingIncludes`	List of protected mappings. Regular expressions are allowed.
	Allowed value: `path?[name=value,#]`
	path is a URL path part. It may contain asterisks (`*` and `**`). The single asterisk `*` symbolizes any character sequence up to the slash (/), while `**` applies to the entire path.
	Example
	`/folder1/folder2 matches against /folder1/*, while /folder1/folder2/folder3 does not.`
	`/folder1/folder2 matches against /folder1/**, as well as /folder1/folder2/folder3.`
	`?[..]` block is optional. Query parameters can be specified inside the block. Parameters are comma separated. The special character # means that the specified parameters are a subset of those from the request; omitting # requires the request parameters to exactly match the specified parameters.
	Parameters may contain only `name`. The match will be done against `name` only, or against `name=value` (that is, both `name` and `value`). A parameter can take multiple values. In this case, the match test will pass if any of the specified parameter values match the corresponding parameter value from the request.
	Example
	`/file1[size=1|2] matches against /file1?size=2, but not against /file1?size=2&author=admin`
	`/file1[size=1|2,name=file1,#] matches against /file1?size=2 and /file1?size=2&author=admin,but not against /file1?size=3`

Table 63-4 (Cont.) id="config"

Property	Description
`protectedMappingInclude s` *(continued)*	To make custom REST resources in an application available through remote Satellite Server, specify the following value: `/ContentServer?[pagename=rest/<path toCSElement>,#]` **Example** `/ContentServer?[pagename=rest/sample/ recommendation,#]` for custom REST resources in Creating REST Resources for WebCenter Sites and Satellite Server: Example.
`proxyTickets`	Specifies whether to proxy tickets. Set this property to `false` for the last server in the call chain for optimal performance. Set this property to `true` if you have to call another CAS-protected application from this application on behalf of the currently logged-in user. This results in the ability to call the following method: `SSO.getSSOSession().getTicket(String service, String username, String password)` **Default value:** `true`
`useMultiTickets`	Specifies whether to use multitickets. **Default value:** `true`

63.1.4.2 Query Parameters Processed by SSO Filter

This table describes query parameters that are processed by SSO filter.

Table 63-5   Query Parameters Processed by SSO Filter

Property Name	Description
`ticket`	Used to verify user identity. Can be used only during some limited period of time for one resource and only once. **Type:** `<query parameter>` **Value:** `<random string>`
`multiticket`	Used to verify user identity. Can be used only during some limited period, multiple times for any resource. **Type:** `<query parameter>` **Value:** `<random string>`

Table 63-5   (Cont.) Query Parameters Processed by SSO Filter

Property Name	Description
`gateway`	If this property is set to `true`, the request for public pages will be redirected to CAS. If the ticket granting cookie is present, then the user will be implicitly authenticated; if not, the user will be redirected back to the original location. This is primarily to allow implicit authentication if the user is logged in to another application. **Type:** `<query parameter>` **Value:** `true ｜ false`
`auth-redirect`	Used to specify the default behavior on unauthenticated access to protected pages. If this property is set to `true`, the user will be redirected to the CAS login page; if `false`, a `403` error will be presented. **Type:** `<Pragma HTTP header>` **Value:** `true ｜ false`

63.2 About Configuring CAS

Here are some sources for information on CAS clustering.

- For information about CAS architecture, use the following link:

  http://www.jasig.org/cas/about

- For information about configuring CAS clustering during the WebCenter Sites installation, see Setting Up a CAS Cluster in *Oracle Fusion Middleware Installing and Configuring Oracle WebCenter Sites*.

- For information about configuring CAS with LDAP providers, use the following link:

  http://www.jasig.org/cas/server-deployment/authentication-handler

63.3 REST Authorization

REST authorization is the process of granting privileges to perform REST operations on applications' resources (which map to objects in WebCenter Sites). REST authorization uses the deny everything by default model. A privilege is denied when it is not explicitly granted to a particular group.

Topics:

- Security Model

- Use of the Security Model to Access REST Resources

- About Configuring REST Security

- Privilege Resolution Algorithm

63.3.1 Security Model

The WEM security model is based on objects, groups, and actions. Security must be configured per object type in the Admin interface. Objects of a given type are accessible to a user only if the user belongs to at least one group with privileges to perform specified actions on the objects of the given type.

Figure 63-2   Add New Security Configuration

- **Object** is a generic term that refers to any entity in the WEM Framework such as a site, a user, or an asset. Protected objects are of the following types:

    – Asset Type

    – Asset

    – Index

    – Site

    – Role

    – User

> > – User Locale
> >
> > – ACL
> >
> > – Application
>
> - **Security groups** are used to gather users for the purpose of managing their permissions (to operate on objects) simultaneously.
>
> - An **action** is a security privilege: LIST, READ, UPDATE, CREATE, DELETE. LIST provides GET permission on services that list objects (such as /types), whereas READ provides GET permission on services that retrieve individual objects in full detail (such as /types/{assettype}).
>
>   Privileges are assigned to groups to operate on allowed objects. Some objects, such as ACLs, are read-only (they can be created directly in WebCenter Sites, but not over REST).

A security configuration is an array, such as shown above, which specifies:

- The protected object type and object(s)

- Groups that are able to access the objects

- Actions that groups (and their members) can perform on the objects

For more information about possible security configurations and the Web Experience Management Framework, see the *Oracle Fusion Middleware Administering Oracle WebCenter Sites*.

63.3.2 Use of the Security Model to Access REST Resources

Object types and objects in WebCenter Sites map to REST resources in the WEM Framework. For example, the Asset Type object maps to:

- <BaseURI>/types/ resource (which lists all asset types in the system)

- <BaseURI>/types/<assettype> resource (which displays information about the selected asset type), and so on

Actions in WebCenter Sites map to REST methods in the WEM Framework. For example, granting the READ privilege to group Editor to operate on asset type Content_C gives users in the Editor group permission to use GET and HEAD methods on the REST resource /types/Content_C.

- The LIST action allows group members to use GET methods on REST resources.

- The READ action allows group members to use GET and HEAD methods on REST resources.

- The UPDATE action allows group members to use POST methods on REST resources.

- The CREATE action allows group members to use PUT methods on REST resources.

- The DELETE action allows group members to use DELETE methods on REST resources.

For comprehensive information, see the *Oracle Fusion Middleware REST API Resource Reference for Oracle WebCenter Sites*.

63.3.3 About Configuring REST Security

See Using REST Security in *Oracle Fusion Middleware Administering Oracle WebCenter Sites*.

63.3.4 Privilege Resolution Algorithm

When configuring a security privilege, specify that the privilege applies to all objects of a certain type or a single object of a certain type. For example, granting the privilege to UPDATE (POST) any site allows users in the group to modify the details of all sites in the WEM Framework.

The Asset object type requires you to specify the site to which the security setting applies, as assets are always accessed from a particular site. The AssetType object can be extended by specifying a subtype, which is used to make the security configuration more granular. For example, setting the DELETE privilege on asset type Content_C allows a DELETE request to be performed on the REST resource /types/Content_C (that is, to delete the Content_C asset type from the system).

Because privileges can be granted only to groups, a user's total privileges are not obvious until they are computed across all of the user's groups. The WEM Framework provides a privilege resolution algorithm. Its basic steps are listed below:

1. REST finds the groups in which the user has membership.

2. REST determines which groups can perform which REST operations on which REST resources. If site or subtype is specified, each is taken into account.

3. REST compares the results of steps 1 and 2. Access is granted if at least one of the groups from step 1 is in the list of groups from step 2. Otherwise, access is denied.

63.4 Management of Assets Over REST

Sample code that illustrates how you can manage assets with the WebCenter Sites REST API is available in your WebCenter Sites installation directory.

See the following locations:

```
Misc/Samples/WEM Samples/REST API samples/Basic Assets/com/fatwire/rest/samples/
basic/
Misc/Samples/WEM Samples/REST API samples/Basic Assets/com/fatwire/rest/samples/
flex/
```

The subfolders basic and flex each contain the following set of files:

*   CreateAsset.java

*   DeleteAsset.java

*   ReadAsset.java

*   UpdateAsset.java.

The code is richly documented with step-by-step instructions.

64

Introducing Customizable Single Sign-On Facility in WEM Framework

WEM Framework's authentication includes a customization layer called the Oracle Customizable Single Sign-On (*CSSO*) facility. You can use the CSSO facility's authentication extensions to create a custom SSO solution, without directly modifying the CAS configuration. When you implement the login behavior, the Spring configuration injects these extensions into the CAS configuration.

Topics:

- About Customizing Login Behavior for the WEM Framework

- About Components of the Default CSSO Implementation

- Configuring and Deploying Custom SSO Behavior

- Running the CSSO Sample Implementation

64.1 About Customizing Login Behavior for the WEM Framework

You need to extend the CSSO facility's pre-packaged classes to implement a custom SSO solution. The CSSO facility's default Spring configuration file identifies the classes to Spring for instantiation.

When you customize WEM SSO, you can use a different login screen, add credentials other than a user name/password pair, or use an external authentication authority to authenticate WebCenter Sites users. A custom SSO implementation consists of:

- Three Java classes (which extend the default classes)

- A configuration file that exposes the new classes to the framework

The default CSSO classes defer all credential discovery and authentication to the standard WEM SSO implementation. These classes are instantiated by the `customdefaultWEMSSObean.xml` Spring configuration file. Extending the default CSSO classes enables you to define methods which specify the behavior of your custom SSO solution. For example, you can create a different authentication for browser access, REST, or thick client authentication or both. When you extend the default CSSO classes, you must create a custom Spring configuration file that identifies the custom classes and exposes them to the WEM Framework.

The CSSO facility provides a complete SSO sample (including Java source files) that replaces the default WEM login behavior with custom login behavior. The sample SSO implementation demonstrates two different types of authentication: user name/password pair (with an additional domain field) and external user identifier. The external identifier maps a user authenticated by an external authentication authority to a WebCenter Sites system user.

The rest of this chapter provides information about the default components of the CSSO facility and instructions on implementing a custom SSO solution. To see an example of a custom SSO solution, the end of this chapter provides information about the CSSO sample, and instructions for running the sample.

64.2 About Components of the Default CSSO Implementation

Your starting point for customizing your SSO implementation is the default components of the CSSO facility. These components are the default classes that you extend to create your SSO solution and a Spring configuration file that instantiates these classes.

The `com.fatwire.wem.sso.cas.custom.basis` package (shown in the table below) contains the default classes that are included in the CSSO facility. The default Spring configuration file (`customdefaultWEMSSObeans.xml`) instantiates these classes to implement the default WEM login behavior.

> **Note:**
>
> The CSSO facility provides a complete SSO sample that replaces the default WEM login behavior with custom login behavior. See Running the CSSO Sample Implementation.

Table 64-1   com.fatwire.wem.sso.cas.custom.basis

Class	Description
CustomAuthenticator.java	Implements the CustomAuthentication interface. This class controls the behavior of the login sequence and handles authentication requests. By default, it returns to the WEM Framework to complete the authentication by displaying the standard WEM login form.
CustomConfiguration.java	Provides access to the properties that are set in the default Spring configuration file. You can extend this class when additional properties are required for a custom SSO implementation.
CustomCredentials.java	Provides a standard set of credential values for custom authentication. You can extend this class when additional attributes are needed for a custom SSO implementation.

The `com.fatwire.wem.sso.cas.custom.interfaces` package (shown in the table below) defines the custom authentication interfaces.

Table 64-2   com.fatwire.wem.sso.cas.custom.interfaces

Class	Description
CustomAuthenticator.java	Defines the interfaces that must be implemented by any custom SSO solution.
CustomRestCodec.java	Defines the interfaces that must be implemented to encode and decode a custom REST authentication token that is not user name/password based.

64.3 Configuring and Deploying Custom SSO Behavior

You begin configuring and deploying your custom SSO by extending the CSSO facility's default classes. Then, you identify the new Java classes to Spring by creating a custom Spring configuration file. This file instantiates the classes, exposing them to the CSSO framework.

To configure and deploy custom SSO behavior:

1. Extend the default CSSO classes: `CustomAuthenticator.java`, `CustomConfiguration.java`, and `CustomCredentials.java` (contained within the `com.fatwire.wem.sso.cas.custom.basis` package):

   a. Create Java classes that extend the default CSSO classes.

   b. Package the Java classes you created in a `jar` file, then place the `jar` file in the classpath of the CAS servlet (in `cas/WEB-INF/lib`).

2. Set the `username` and `password` properties in the `customResolverCredential.xml` file, located in the `spring-configuration` folder (in `cas/WEB-INF`). For instructions, see Settings Resolver Credentials.

3. Identify your new Java classes to Spring for instantiation:

   a. Create a Spring configuration file that contains all the custom class names and properties for your SSO implementation.

   b. Place the custom Spring configuration file in the `spring-configuration` folder (in `cas/WEB-INF/`).

   c. Remove the `.xml` extension from the default Spring configuration file (`customDefaultWEMSSObeans.xml`).

4. If an external authentication authority is used to authenticate a user, map the external user identifier to the appropriate WebCenter Sites system user name, unique identifier, and ACLs. For instructions, see Mapping External User Identifiers to WebCenter Sites Credentials.

5. Restart the CAS web application.

64.3.1 About Extending the Default CSSO Classes

An SSO implementation is a set of called methods that are specified in the default CSSO classes `CustomAuthenticator.java`, `CustomConfiguration.java`, and `CustomCredentials.java`. To replace the default WEM login behavior with custom behavior, you must create Java classes that extend the default CSSO classes. By extending the CSSO classes, the methods specified in the default CSSO classes are replaced by the methods specified in the custom classes for the functionality you want to change.

The three classes (located in the `com.fatwire.wem.sso.cas.custom.basis` package) that must be extended to implement a custom SSO solution are:

- **CustomConfiguration.java**: Provides access to the externally defined properties that are specified in the default Spring configuration file. By default, this class exists only as a placeholder for injecting properties into the SSO configuration from the Spring configuration file. Extend this class to include additional

properties, such as URLs or other configuration information, that are specific to your custom SSO implementation.

- **CustomCredentials.java**: Provides a standard set of credential values for custom authentication. This class is built and populated by the `web-flow` handler or the custom REST authenticator. By default, this class defines the standard `UsernamePasswordCredentials` object (provided by CAS), which collects all information required to complete user authentication in the following properties; `username`, `userId`, and `currentACL`. The values of these properties populate the attributes map used by the authenticator (`CustomAuthenticator.java`), to perform the actual user authentication.

  Extend this class to require additional credentials for your custom SSO solution. For an example of how this class passes user information to the authenticator to complete user authentication, refer to the code of the sample CSSO class `SampleCredentials.java` (located in the `Misc/Samples/WEM/Samples/CustomizableSSO/lib` folder).

- **CustomAuthenticator.java**: Implements the `CustomAuthentication` interface. This class controls the behavior of the login sequence and handles authentication requests. By default, it returns to the WEM Framework to complete the authentication by displaying the standard WEM login form.

  > **Note:**
  >
  > The default `CustomAuthenticator.java` class is the most important class because it contains all the authentication methods for an SSO implementation.

  All authentication decisions and CAS web-flow actions are directed to this class for action. CAS web-flow performs several steps, one of which invokes the `performLoginAction` method. This method displays a login form or communicates with an external authentication authority.

  This class also defines the static method `callCsResolverPage` which maps an external user to a WebCenter Sites user. If your custom SSO implementation uses an external authentication authority to authenticate users, the `callCsResolverPage` method must define the unique name for the CSSO authenticator. See Mapping External User Identifiers to WebCenter Sites Credentials.

  The following is a complete interface description of the methods this class implements:

```
static final int SUCCESS  = 0;
static final int GOTOWEM  = 1;
static final int FAILURE  = 2;
static final int REDIRECT = 3;
static final int ERROR    = 4;
static final int REPEAT   = 5;


/**
 * Called from UserAuthentication handler to check for alternate
 * credentials and validate appropriately.
 * @param userCredentials
 * @return
 */
public int authenticate(com.fatwire.wem.sso.cas.custom.basis.CustomCredentials
userCredentials);
```

```
/**
 * Called from CSAuthenticationHandler to check for REST user
 * credentials and validate appropriately.
 */
public int authenticateRest(UsernamePasswordCredentials restCredentials);

/**
 * Called from CSAuthenticationHandler to check is username/password
 * combination is detected.
 */
public boolean checkRestCredentials(String token);

/**
 * Called from CSAttributeDAO to check for encoded credentials and
 * if so then return the correct username for DAO processing.
 * @param username
 * @return
 */
public String resolveRestUsername(String username);

/**
 * Called from LoginViewAction to handle login view processing. This
 * method allows the calling of internal CAS methods.
 * @param context
 * @param userAuthentication
 * @param centralAuthenticationService
 * @return
 */
public int performLoginAction(RequestContext context,
CustomAuthentication userAuthentication,
CentralAuthenticationService centralAuthenticationService);

/**
 * Called from casLogoutView to perform sign in cleanup
 * @param request
 * @param response
 */
public void performLogoutAction(HttpServletRequest request, HttpServletResponse
response);
```

64.3.2 Settings Resolver Credentials

The `customResolverCredentials.xml` file, located in the `spring-configuration` folder (in `cas/WEB-INF`), defines the resolver credentials externally, so the credentials are encrypted independent from the custom SSO bean definitions. If an external authentication authority is used to authenticate users, you must set the `username` and `password` properties in the `customResolverCredential.xml` file and then reference this file in the Spring configuration file.

To set resolver credentials:

1. Open for editing the `customResolverCredentials.xml` file, located in the `spring-configuration` folder (in `cas/WEB-INF`). The file looks as follows:

```
<?xml version="1.0" encoding="UTF-8" ?>
<beans xmlns="http://www.springframework.org/schema/beans" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance" xmlns:p="http://www.springframework.org/
schema/p" xsi:schemaLocation="http://www.springframework.org/schema/beans http://
www.springframework.org/schema/beans/spring-beans-2.0.xsd">
```

```
   <description>This defines the SitesCS username/password credentials needed for
identity resolution</description>
<bean id="resolverCredential"
class="com.fatwire.security.common.SecurityCredential">
  <property name="username" value="theUsername" />
  <property name="password" value="theUserpassword" />
  <property name="csfKeyname" value="passwordkey" />
  </bean>
  </beans>
```

2. Set the `username` and `password` properties.

   - `username` – If an external authentication authority is used to authenticate a user, this property must specify the user name of a WebCenter Sites user who has permissions to read the `SystemAttr` table. This user name is used when the `customCsResolver` page queries the WebCenter Sites database to resolve an external user identifier into a registered WebCenter Sites user.

   - `password` – If an external authentication authority is used to authenticate a user, this property must specify the password of the user identified by the `username` property.

   For example:

   ```
   <property name="username" value="fwadmin" />
   <property name="password" value="xceladmin" />
   ```

   See Mapping External User Identifiers to WebCenter Sites Credentials.

64.3.3 About Identifying Your Java Classes to Spring for Instantiation

All customization settings for an SSO implementation are specified in a single Spring configuration file, located in the `spring-configuration` folder (in `cas/WEB-INF`).

This section includes the following topics:

- About Creating a Spring Configuration File

- About Placing Your Spring Configuration File

64.3.3.1 About Creating a Spring Configuration File

The classes and properties for the default SSO implementation are defined by the Spring configuration file `customDefaultWEMSSObeans.xml`, which is located in the `spring-configuration` folder (in `cas/WEB-INF`). When customizing CSSO, either create a new Spring configuration file or customize the classes and properties referenced in the default Spring configuration file. The rest of this section focuses on the second option.

The default Spring configuration file contains several bean identifiers that reference the classes and properties required for the default SSO implementation. The `customUserConfiguration` bean references the `CustomConfiguration.java` class and the `customUserAuthenticator` bean references the `CustomAuthenticator.java` class. These classes are instantiated by the Spring configuration file, which uses them to create the persistent objects for the SSO implementation's authentication process. To create a custom SSO solution, you must reference your custom Java classes within these beans.

> **Note:**
>
> The `CustomCredentials.java` class is **not** referenced by the Spring configuration file. Instead, you provide the code that instantiates this object in the `performLoginAction` method, defined in the default CSSO `CustomAuthenticator.java` class. This method creates a custom credentials object for every login request and passes it into CAS for authentication.

The `customUserConfiguration` bean also identifies the configuration properties which supply system information to the default SSO implementation. These properties are set with values of the environment on which you are deploying the SSO implementation. When you customize the Spring configuration file, you must modify the values of the properties to match the custom SSO implementation's environment, or include additional properties required by the custom SSO implementation.

Extending the `CustomConfiguration.java` class enables you to define additional properties in the Spring configuration file's `customUserConfiguration` bean. For example, if you created a JSP file that provides a custom login form for your SSO implementation, create a property that specifies the location of the JSP file by extending the `CustomConfiguration.java` class.

The rest of this section analyzes the classes and properties that are referenced in the default Spring configuration file (`customDefaultWEMSSObean.xml`).

The Default Spring Configuration File

```
<?xml version="1.0" encoding="UTF-8" ?>
<beans xmlns="http://www.springframework.org/schema/beans"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:flow="http://www.springframework.org/schema/webflow-config"
                xmlns:p="http://www.springframework.org/schema/p"
                xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
                http://www.springframework.org/schema/webflow-config http://
www.springframework.org/schema/webflow-config/spring-webflow-config-1.0.xsd">
<!-- Custom SSO Bean definitions. This file defines either the default CAS/SSO
configuration or a special user implementation. No other CAS configuration files are
modified for a custom implementation    -->
<!-- This file defines the resolver credentials externally so the credentials can
be encrypted independent from the custom SSO bean definitions    -->
 <import resource="customResolverCredential.xml" />
 <!-- This bean is never modified. It defines the web-flow controller which always
passes control into the custom authenticator    -->
 <bean id="customUserLoginAction"
class="com.fatwire.wem.sso.cas.web.CustomLoginViewAction"
p:centralAuthenticationService-ref="centralAuthenticationService"
p:customAuthentication-ref="customUserAuthenticator" />
 <!-- This bean is usually not modified. Override it when there needs to be a
custom encoding for information passed between the web-flow and any external
component    -->
 <bean id="customRestCoder"
class="com.fatwire.wem.sso.cas.custom.basis.CustomRestTokenCoding" />
 <!-- Modify this bean with a custom configuration implementation class when
additional parameters are needed for a custom implementation    -->
 <bean id="customUserConfiguration"
class="com.fatwire.wem.sso.cas.custom.basis.CustomConfiguration"
p:casLoginUrl="@CSConnectPrefix@://@hostname@:@portnumber@/cas/login"
p:resolverUrl="@CSConnectPrefix@://@hostname@:@portnumber@/@context-path@/custom/
```

```
customCsResolver.jsp" p:resolverCredential-ref="resolverCredential"
p:traceFlag="false" />
 <!--  Modify this bean with a customAuthentication class for a custom
implementation.   -->
 <bean id="customUserAuthenticator"
class="com.fatwire.wem.sso.cas.custom.basis.CustomAuthenticator"
p:customConfiguration-ref="customUserConfiguration"
p:customRestCoder-ref="customRestCoder" />
</beans>
```

Analyzing the Default Spring Configuration File

- The `customResolverCredentials.xml` file in which you specified the resolver credentials externally is imported. See Settings Resolver Credentials.

- The `customUserConfiguration` bean, which references the default CSSO `customConfiguration.java` class. See About Extending the Default CSSO Classes.

- The `resolverURL` property and the `traceflag` property are referenced.

    - `resolverURL`: If an external authentication authority is used to authenticate a user, this property must specify the full URL to the `customCsResolver` page, located on WebCenter Sites. The `customCsResolver` page obtains a user's external identifier and queries the WebCenter Sites database to retrieve the user's WebCenter Sites credentials. The domain and port number specified in this property must be modified if the values specified are different from the WebCenter Sites installation.

        See Mapping External User Identifiers to WebCenter Sites Credentials.

    - `traceflag`: This property specifies whether the trace log, which provides information about the custom SSO layer, is enabled. This property can be set to either `true` or `false`.

- The `customUserAuthenticator` bean references the default CSSO `CustomAuthenticator.java` class. See About Extending the Default CSSO Classes.

64.3.3.2 About Placing Your Spring Configuration File

The default Spring configuration file, which specifies the classes and properties for the default WEM login behavior, is located in the `spring-configuration` folder (in `cas/WEB-INF`). Placing your own file into the same location requires deactivating the default file (by removing or changing the file's `.xml` extension). This is because Spring loads all Spring configuration files contained in the `spring-configuration` folder (in `cas/WEB-INF`) and merges those files into a single configuration. As both the custom and the default files specify the same bean identifiers, only one of the files can be recognized by the Spring configuration. Duplicate bean identifiers result in initialization failure.

Note:

Avoid deleting `customDefaultWEMSSObeans.xml`. Instead, remove or change the file's `.xml` extension. This way you can restore the file to return to using the default WEM login screen.

64.3.4 Mapping External User Identifiers to WebCenter Sites Credentials

The CSSO facility enables you to use an external authentication authority to authenticate WebCenter Sites users. When the external authentication authority validates the user's credentials, it associates a unique external identifier with that user. To complete WEM authentication, the user's external identifier must be mapped to the corresponding WebCenter Sites system user name, unique identifier, and ACLs by using the method `callCsResolverPage` (defined as a static method in the default CSSO class `CustomAuthenticator.java`).

To map an external identifier to a WebCenter Sites system user, ensure you have set the external authentication properties in the `customResolverCredentials.xml` file (see Settings Resolver Credentials). To implement mapping from an external identifier to the appropriate WebCenter Sites system credentials, do the following:

To implement mapping:

1. Define a unique CSSO authenticator name for the external authentication authority of your custom SSO implementation in the `callCsResolverPage` method (defined in your extended `CustomAuthenticator.java` class).

   For example, the following `callCsResolverPage` method (defined in the Sample CSSO class `SampleAutheticator.java`) defines `samplesso` as the unique authenticator name:

   ```
   Map<String,String>csTokens=callCsResolverPage(externalUserId
   , "samplesso")
   ```

2. Access the Admin interface as a general administrator (for example, `fwadmin/xceladmin`).

3. Under the **Admin** node, expand the **User Access Management** node and double-click **User**.

4. Select the user whose external identifier you want to map to WebCenter Sites credentials:

   a. In the **Enter User Name** field, enter the name of the user.

   b. In the **Select Operation** section, select the **Modify User Attributes** option.

   c. Click **OK**.

   The Modify User form opens.

   Figure 64-1 Modify User Form

5. In the **User Name** column, click the name of the user whose external identifier you want to map to WebCenter Sites credentials.

   The User Attribute form opens.

Figure 64-2   User Attribute Form

6. In the form, fill in the fields:

   • In the **Attribute Name** field, enter the unique CSSO authenticator name (the name used to identify the external authentication authority). This name must match the unique name of the CSSO authenticator defined in the `callCsResolverPage` method (in step 1).

   • In the **Attribute Values** field, enter the user's external identifier provided by the external authentication authority.

7. Click **Modify** to store the new attribute and value in the WebCenter Sites `SystemUserAttr` database table.

8. Repeat steps 3 through 7 for all users associated with an external identifier.

Analysis of the Mapping Process

When the `callCsResolverPage` method is called to map an external identifier to a WebCenter Sites system user, it defines the unique CSSO authenticator name for your custom SSO implementation. The method uses the external identifier and the unique CSSO authenticator name to map the external user to the WebCenter Sites system user. This map contains the following items, which are placed in the associated properties of the `CustomCredentials` object:

• `username`: The user's WebCenter Sites user name.

• `currentUser`: The user's WebCenter Sites unique identifier.

• `currentACL`: The user's ACLs.

The `CustomCredentials` object passes the `username`, `currentUser`, and `currentACL` values to the `authenticate` method, defined in the `CustomAuthenticator.java` class. The `authenticate` method uses these values to build the response map, which identifies the WebCenter Sites user.

64.3.5 Restarting the CAS Web Application

• To deploy your custom SSO implementation, restart the CAS web application.

   Once CAS has been restarted, it uses the classes defined in the custom Spring configuration file, located in the `spring-configuration` folder (in `cas/WEB-INF`) to provide the custom login behavior.

64.4 Running the CSSO Sample Implementation

Try out CSSO's sample SSO implementation (including Java source files) to replace the default WEM login behavior with custom login behavior. This sample includes the standard user name and password fields, an additional field for a user to specify a domain name, and a field for an external user identifier. You will find two different

types of authentication in this implementation, user name/password pair (with an additional domain field) and user authentication through an external authentication authority.

> **Note:**
>
> The CSSO sample does not enforce any validation rules that apply to the fields on the login form. Fields are not checked for completeness and incorrect values are not reported. If authentication fails, the form is re-displayed without comment. When implementing this form in a production environment, ensure that all rules are enforced with suitable diagnostic messages if an error occurs.

For information about all the sample components included in the CSSO facility, see Sample CSSO Components.

To run the sample SSO implementation:

1. Deploy the `customizable-sso-1.0.jar` (`Misc/Samples/WEM Samples/CustomizableSSO`) by placing it in the CAS classpath (`cas/WEB-INF/lib` folder). This file contains the sample CSSO classes.

2. Create a `fatwire` folder in the CAS web application context folder. Copy the `SampleLoginform.jsp` file into the `fatwire` folder.

3. Identify the classes contained in the `customizable-sso-1.0.jar` file to Spring for instantiation:

   a. Copy the `customSampleSSObeans.xml` configuration file into the `spring-configuration` folder.

   b. Modify the properties in the `customSampleSSObeans.xml` file to match your operation environment.

   c. Remove the `.xml` extension from the `customDefaultWEMSSObeans.xml` configuration file's name, located in the `spring-configuration` folder.

4. To use the external identifier credentials to validate users, define the mapping relationship between the external user identifier and the user's WebCenter Sites system credentials by adding the appropriate entry to the `SystemUserAttr` table.

   For instructions, see Mapping External User Identifiers to WebCenter Sites Credentials.

5. Restart the CAS web application.

   The sample login form opens.

Figure 64-3   Sample CAS Login Form

64.4.1 Sample CSSO Classes

The CSSO sample contains three Java classes which extend the default CSSO classes, providing the methods for the sample SSO implementation's login behavior:

- **SampleConfiguration.java**: This class extends the default CSSO `CustomConfiguration.java` class to include a domain property (`sampleDomain`) which will be validated by an external authentication authority when a user provides a value for this field on the login form. The `sampleDomain` property is injected into the CSSO configuration by Spring.

  This class also includes the `sampleFormURL` property which defines the sample login form that is called to retrieve a user's credentials. Standard and custom properties for this class are supplied through the sample Spring configuration file.

- **SampleCredentials.java**: This class extends the default CSSO `CustomCredentials.java` class and collects all information required to complete user authentication. The `SampleAuthenticator` class uses the `UsernamePasswordCredentials` object when a user supplies a user name and password on the login form. If a user supplies an external identifier on the login form instead of user name and password credentials, the `SampleCredentials` object is created to provide that information to the authenticator (in this example, sample SSO class `SampleAuthenticator.java`).

  In CAS, the type of credentials object that is created controls which authenticator is used (either standard or custom). If user name and password credentials are supplied on the login form, the standard WEM user name and password authenticator is used automatically. If an external identifier is supplied on the login form, the custom authenticator is called to authenticate the `SampleCredentials` object.

- **SampleAuthenticator.java**: This class extends the default CSSO `CustomAuthenticator.java` class and contains all the authentication methods that are called by the CSSO framework. When the sample is deployed, all authentication decisions and web-flow actions, during CAS authentication, are directed to this class for action.

  The `performLoginAction` method (extended by this class) displays the sample login form. When a user submits his credentials on the form, CAS returns to this method to process the input fields. Depending on the credentials that require verification, the method creates either a `UsernamePasswordCredentials` object or a `SampleCredentials` object, populated with the user's assigned credentials. The credentials object is then inserted into the CAS context (provided by CAS) and a TGT is requested. The TGT request triggers authentication of the

credentials object. If authentication is denied, a ticket exception results in the login form being redisplayed. If the authentication is successful, the next action in the web-flow occurs. For example, acquire a ticket, append the ticket to the original service URL (the WebCenter Sites URL), and redirect back to the original service.

There are two authentication methods in this class. One handles authentication using `SampleCredentials` and the other authenticates REST requests, which are usually user name/password based. The sample introduces the `sampleDomain` value as a new value to be authenticated. In this case, the `performLoginAction` method encodes the user name, password, and `sampleDomain` values provided by the user and passes the encoded values to the `UsernamePasswordCredentials` object. The default WEM authentication handler detects the `sampleDomain` value and passes that credential to the `authenticationRest` method. This method decodes the `sampleDomain` value from the other values and verifies that the correct domain has been specified. Authentication fails if the value is incorrect. If the value is correct, this method encodes the user name and password back into the credentials object, and the default WEM authentication handler validates the user name and password.

64.4.2 Sample Spring Configuration File

The classes and properties for the sample SSO implementation are defined by the sample Spring configuration file `customSampleSSObeans.xml` (located in `Misc/Samples/WEM Samples/CustomizableSSO/src/main/webapp/WEB-INF/spring-configuration`).

This section includes the following topics:

- Analysis of the Sample Spring Configuration File

- Placing the Sample Spring Configuration File

64.4.2.1 Analysis of the Sample Spring Configuration File

The sample Spring configuration file contains the same bean identifiers as the default Spring configuration file. However, the property values are modified to implement the sample login behavior. For example, the `customUserConfiguration` bean references the `SampleConfiguration.java` class and the `customUserAuthenticator` bean references the `SampleAuthenticator.java` class.

The `customUserConfiguration` bean also identifies the configuration properties which supply system information to the sample SSO implementation. For example, since the `SampleLoginForm.jsp` file provides the browser form that is used by the sample to obtain a user's credentials, the `SampleConfiguration.java` class is extended to include the `sampleFormURL` property. This property specifies the full URL of the login page for the sample SSO implementation. The domain name and port number match the CAS server installation, and the path points to where this page was placed during set up.

The following is the sample Spring configuration file's code. For more information about the properties referenced by this file, see Analyzing the Default Spring Configuration File in About Creating a Spring Configuration File.

The Sample Spring Configuration File

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
            xmlns:flow="http://www.springframework.org/schema/webflow-config"
        xmlns:p="http://www.springframework.org/schema/p"
        xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
http://www.springframework.org/schema/webflow-config http://www.springframework.org/
schema/webflow-config/spring-webflow-config-1.0.xsd">

<!-- Custom SSO Bean definitions. This file defines either the default CAS/SSO
configuration or a special user implementation. No other CAS configuration files are
modified for a custom implementation -->

<!-- This file defines the resolver credentials externally so the credentials can
    be encrypted independent from the custom SSO bean definitions -->
<import resource="customResolverCredential.xml" />

<!-- This bean is never modified. It defines the web-flow controller which always
    passes control into the custom authenticator -->
<bean id="customUserLoginAction"
class="com.fatwire.wem.sso.cas.web.CustomLoginViewAction"
        p:centralAuthenticationService-ref="centralAuthenticationService"
        p:customAuthentication-ref="customUserAuthenticator"
        />
<!-- This bean is usually not modified. Override it when there needs to be a
    custom encoding for information passed between the web-flow and any external
    component -->
<bean id="customRestCoder" class="com.fatwire.wem.sso.cas.custom.basis.
        CustomRestTokenCoding"
        />
<!-- Modify this bean with a custom configuration class when additional parameters
are needed for a custom implementation -->
<bean id="customUserConfiguration"
class="com.fatwire.wem.sso.cas.sample.SampleConfiguration"
        p:casLoginUrl="http://localhost:8080/cas/login"
        p:resolverUrl="http://localhost:8080/cs/custom/customCsResolver.jsp"
        p:resolverCredential-ref="resolverCredential"
        p:traceFlag="false"
        p:sampleDomain="mydomain"
        p:sampleFormUrl="http://localhost:8080/cas/fatwire/SampleLoginForm.jsp"
        />
<!-- Modify this bean with a customAuthentication class for a custom implementation.
-->
<bean id="customUserAuthenticator"
class="com.fatwire.wem.sso.cas.sample.SampleAuthenticator"
        p:customConfiguration-ref="customUserConfiguration"
        p:customRestCoder-ref="customRestCoder"
        />

</beans>
```

64.4.2.2 Placing the Sample Spring Configuration File

To instantiate the sample classes, place the sample Spring configuration file in the
spring-configuration folder (in cas/WEB-INF) and remove the .xml extension
from the default Spring configuration file.

For more information, see About Placing Your Spring Configuration File.

64.4.3 Sample CSSO Components

The sample CSSO implementation's components are located in the `/WEM Samples/` `CustomizableSSO` folder. The folders described in the table below are included with the sample CSSO implementation.

Table 64-3   Sample CSSO Components

Folder	Description
Misc/Samples/WEM Samples/ CustomizableSSO	Contains the `customizable-sso-1.0.jar` file. This `jar` file provides the classes of the executable code for the sample. To deploy the sample SSO implementation, place this `jar` file in the CAS classpath (`cas/WEB-INF/lib` folder).
Misc/Samples/WEM Samples/ CustomizableSSO/lib	Contains all the third-party `jar` files required to compile the Java source files for the sample SSO implementation.
Misc/Samples/WEM Samples/ CustomizableSSO/src/main/dist	Contains Word documents that explain the individual source components and operations of the sample implementation. **Note:** We recommend reviewing these documents before viewing the sample's source code.
Misc/Samples/WEM Samples/ CustomizableSSO/src/main/java	The root folder for the Java source files.
Misc/Samples/WEM Samples/ CustomizableSSO/src/main/webapp/ fatwire	Contains `SampleLoginForm.jsp`. The JSP provides the browser form that is used by the sample to obtain a user's login credentials. Implementing the sample requires creating a `fatwire` folder in the CAS application context folder and copying the `SampleLoginForm.jsp` to that folder.
Misc/Samples/WEM Samples/ CustomizableSSO/src/main/webapp/ WEB-INF/spring-configuration	Contains the sample Spring configuration file `customSampleSSObeans.xml`, which defines the Spring bean definitions required by the sample SSO implementation. This file must be placed in the `spring-configuration` folder (in `cas/WEB-INF`). The file that exists in the `spring-configuration` folder (`customDefaultWEMSSObeans.xml`) must be given an extension other than `.xml` or removed. **Note:** Save a copy of the `customDefaultWEMSSObeans.xml` file so it can be restored when you return to the standard WEM login screen.

65

Buffering in WEM Framework

Asset create, update, and delete operations are much slower than the read operation. Sometimes, you may prefer that these operations occur at a future time with the guarantee of eventual consistency. That is, in case of a delayed (buffered) operation, the WebCenter Sites platform surely receives the change at some finite, undetermined period of time. Although buffering operations are extremely fast, they do not speed up the total time that you need to create, update, and delete assets in the platform.

Topics:

- Architecture of Buffering System

- Using Buffering

65.1 Architecture of Buffering System

The current implementation of buffering subsystem relies on Java Messaging Service (JMS) technology. It has two components, buffering producer and buffering consumer. The producer produces messages and puts them into the Messaging Queue (MQ). The consumer picks messages from MQ and persists them in the platform.

Figure 65-1   Buffering System Architecture
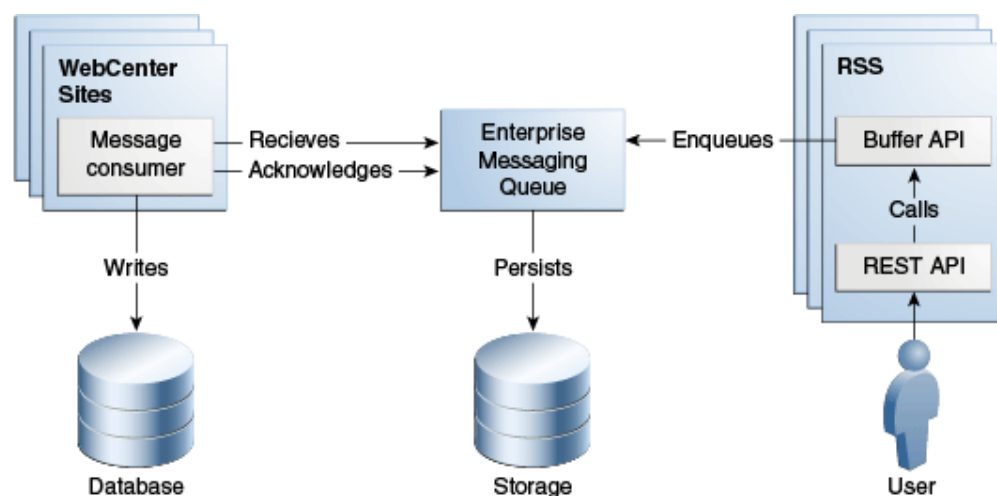
The buffering producer can be used on both WebCenter Sites and Remote Satellite Server, where the asset REST service `<BaseURI>/sites/<sitename>/types/<assettype>/assets/<id>` is available. When used on Remote Satellite Server, the buffering producer does not communicate with WebCenter Sites, which ensures linear scalability of the entire system.

> **Note:**
>
> The buffering consumer is available only on WebCenter Sites. We recommend enabling the buffering consumer only on the primary cluster member. Enabling on multiple cluster members cannot guarantee that the sequence of CRUD operations will be preserved.

65.2 Using Buffering

If you would like to use buffering, go ahead and enable the buffering option in the `BufferingConfig.xml` file.

1. Install the JMS provider if one is not available. (For supported providers, see the *Oracle Fusion Middleware WebCenter Sites Certification Matrix* available from the Oracle Technology Network at http://otn.oracle.com.

2. Configure `BufferingConfig.xml` on WebCenter Sites and optionally on Remote Satellite Server.

```
id="bufferingManager" class=
  "com.fatwire.cs.core.buffering.jms.JmsBufferingManager"
```

Table 65-1   Properties in BufferingConfig.xml

Property name	Description
`jmsConnectionFactory`	Required. Instance of `javax.jms.ConnectionFactory`
`jmsDestination`	Required. Instance of `javax.jms.Destination`
`messageConsumers`	List of `com.fatwire.cs.core.buffering.IMessageConsumer` implementations.

> **Note:**
>
> When you configure `BufferConfig.xml`, add `activemq-all.jar` to the WebCenter Sites web application's classpath (for example, `WEB-INF/lib`).

3. Specify `buffer=true` when invoking the REST asset service `<BaseURI>/sites/<sitename>/types/<assettype>/assets/<id>`.

> **Note:**
>
> Buffering does not return the result of PUT and POST operations in the response. Instead, an empty payload is sent. Developers should be aware of this behavior when coding the client application.

The default `BufferingConfig.xml` file, provided with WebCenter Sites, contains the sample configuration for Apache ActiveMQ. The `BufferingConfig.xml` file is similar for both WebCenter Sites and Remote Satellite Server, except that the list of message consumers for Remote Satellite Server is empty.

66

Registering Applications Manually in WEM Framework

When you register an application manually, you create an asset for the application, create an asset for each of its views, and associate the view assets with the application asset in the Admin interface. You can enable the registration asset types `FW_Application` and `FW_View` on AdminSite.

Topics:

- Registering Applications in WEM Framework

- Reference: Registration Asset Types

66.1 Registering Applications in WEM Framework

Learn how you can register an application and its views by registering the Articles sample application and its view.

See Working with the Articles Sample Application. Articles has a single view of type `iframe`. The same steps apply to JavaScript and HTML views.

To manually register an application and view:

1. Create or get an icon to represent your application. (The icon will be displayed in the applications bar.)

   The Articles sample application uses the `articles.png` image file located in: `/sample app/articles/src/main/webapp/images/`

2. Create a file that specifies the layout of the application in HTML, that is, for each view, create a placeholder element to hold the content rendered by the view.

   For example, `layout.jsp` (for the Articles sample application) contains the following line:

   ```
   <div id="articles" style="float:left;height:100%;width:100%"
     class="wemholder"></div>
   ```

   The view's contents are rendered within the placeholder element when the application is displayed (`layout.app` renders the application's layout; `home.app` renders the view).

   Note:

   When creating the layout file, specify a unique Id for the placeholder element. You will specify the same Id for the Parent Node attribute when creating the view asset. Use `class="wemholder"` for the placeholder elements.

   ---

3. Register the view and application.

a. Log in to the Admin interface as a general administrator, navigate to the AdminSite and expand the **Admin** node, where the `FW_View` and `FW_Application` asset types are enabled.

(We assume you will create the view and application assets in the same session, in which case both assets will be listed on the **History** tab. When creating the application asset, you will select the view asset from the **History** tab and associate it with the application asset. The **History** tab is volatile; it is cleared after the user's session. Assets can be permanently placed on the **Active List** tab. See *Oracle Fusion Middleware Administering Oracle WebCenter Sites*.)

b. Create an instance of the `FW_View` asset type:

Click **New**, select **New FW_View**, and set attributes. (The following figure displays attribute values for the view asset of the Articles sample application.)

Figure 66-1   Attribute Values for the View Asset

Name: Enter a short descriptive name for this view asset.

Parent Node: Enter the `id` of the placeholder element (defined in step 2 of Registering Applications in WEM Framework) that will hold the content rendered by the view.

View Type: Select one of the following options to specify how the view's contents should be rendered in the placeholder:

- `Iframe`: Renders the view in an iframe into the placeholder element

- `IncludeHTML`: Renders HTML into the placeholder element

- `IncludeJavaScript`: Renders JavaScript into the placeholder element

Source URL: Enter the URL that provides contents for the view. For example, Source URL for the Articles sample application takes the following value: `http://localhost:9080/articles-1.0/home.app`

c. Create an instance of the `FW_Application` asset type:

Click **New**, select **New FW_Application**, and set attributes. (The following figure displays attribute values for the application asset of the Articles sample application.)

Figure 66-2   Attribute Values for the Application Asset

Name: Enter a short descriptive name for this application asset.

ToolTip: Enter the text that will be displayed over the application's icon when users hover over the icon.

Icon URL: Enter the URL of the icon that represents the application. The icon will be displayed on the login page and at the top of the WEM interface. For example, the Icon URL for the Articles sample application takes the following value: `http://localhost:9080/articles-1.0/images/articles.png`

Hover Icon URL: Enter the URL of the icon that represents the application when users place mouse over the icon.

Click Icon URL: Enter the URL of the icon that represents the application when users click the icon.

Active Icon URL: Enter the URL of the icon that represents the application when it is in use.

Layout Type: `LayoutRenderer` (the default and only value). Layout Type is used by the UI container to render the application's views by using the application's layout page (specified below in the Layout URL attribute).

Layout URL: Enter the URL of the page that displays the application's layout. The layout page has only HTML placeholder elements (such as div) for placing the view(s).

For example, Layout URL for the Articles sample application takes the value: `"http://localhost:9080/articles-1.0/layout.app"` rather than `".../layout.jsp"`, given the Spring MVC framework.

Related: Associated FW_View: views: Select the view asset created in step 3 of Registering Applications in WEM Framework (click the **History** tab, select the view asset, and click **Add Selected Items**).

66.2 Reference: Registration Asset Types

With the FW_View asset type you can register views of your application. And, you can use the FW_Application asset type to register your application.

Topics:

- FW_View Asset Type

- FW_Application Asset Type

66.2.1 FW_View Asset Type

This asset type is used to register the views of an application. For each view, create an instance of `FW_View`. Attributes of `FW_View` are listed in the table below as they appear in the Admin interface and in the *REST API Bean Reference for Oracle WebCenter Sites*. This asset type is enabled on the site named AdminSite.

Table 66-1 `FW_View` Asset Type Attributes

Attribute: WebCenter Sites Interface	Attribute: REST API	Description
Name	`name`	Short descriptive name for this view asset.
Description	`description`	Description of this view asset.
Parent Node	`parentnode`	ID of the placeholder element in the application's layout file. The placeholder element holds the content rendered by the view. The layout file has only HTML placeholder elements (such as `div`) for placing the views.
View Type	`viewtype`	How the view should be rendered. The following view types are available: `Iframe`: Renders the view in an iframe into the placeholder element `IncludeHTML`: Renders HTML into the placeholder element `IncludeJavaScript`: Renders JavaScript into the placeholder element

Table 66-1   (Cont.) `FW_View` *Asset Type Attributes*

Attribute: WebCenter Sites Interface	Attribute: REST API	Description
Source URL	`sourceurl`	URL that provides content for the view.
JavaScript	`javascriptco ntent`	Required if `IncludeJavaScript` is the view type and Source URL is not specified.
		The content specified by this attribute is included in a script tag if `IncludeJavaScript` is specified as the view type. If `IncludeJavaScript` is the view type, either `Source URL` must be specified, or code must be provided for the `JavaScript` attribute.
Content	`includeconte nt`	Required if `IncludeHTML` is the view type and `Source URL` is not specified. The content specified by this attribute is included in the placeholder element tag if `IncludeHTML` is specified as the view type. If `IncludeHTML` is the view type, either the `Source URL` must be specified or code must be provided for the `Content` attribute.

66.2.2 FW_Application Asset Type

This asset type is used to register the application. The asset type is enabled on AdminSite. Attributes of `FW_Application` are listed in the table below as they appear in the Admin interface and in the .

*Table 66-2   * `FW_Application` *Asset Type Attributes*

Attribute: WebCenter Sites Interface	Attribute: REST API	Description
Name	`name`	Short descriptive name for this application asset.
Description	`description`	Description of this application asset.
Tooltip	`tooltip`	Text that is displayed on the application's icon when users place mouse over the icon.
Icon URL	`iconurl`	URL of the icon that represents the application in the WEM Framework.
Hover Icon URL	`iconurlhover`	URL of the icon that represents the application when users place mouse over the icon.
Click Icon URL	`clickiconurl`	URL of the icon that represents the application when users click the icon.
Active Icon URL	`iconurlactiv e`	URL of the icon that represents the application while it is in use.

Table 66-2   (Cont.) `FW_Application` Asset Type Attributes

Attribute: WebCenter Sites Interface	Attribute: REST API	Description
Layout Type	layouttype	Type of layout. The value is LayoutRenderer. Layout Type is responsible for rendering the application's views by using the application's layout page (specified in the Layout URL attribute).
Layout URL	layouturl	URL of the page where the application's layout is displayed. This page has only HTML placeholder elements (such as div) for placing the views.
Related: Associated FW_Application: extends	parentnode	Parent application which the current application extends.
Related: Associated FW_View: views	views	List of view assets used in this application.

Part XVIII

Customizing Oracle WebCenter Sites

You can customize the Oracle WebCenter Sites: Contributor interface by altering its customizable components. Make best use of the customization methods and supporting code .

You can find the sample code at in the `misc\Samples\UICustomization\` directory of the unbundled WebCenter Sites distribution file.

- Code for customizing search views and the dashboard is provided in the `CustomAttrEditor.zip` file.

- Code for customizing asset forms is provided in the `sample_elements.zip` file.

Customizing the WebCenter Sites Interface:

- Customizing the Tree in the Admin Interface

- About Customizing Components of the Contributor Interface

- Understanding the Contributor Interface Framework and UI Controller

- Customizing the Contributor Interface Dashboard

- Customizing Search Views of the Contributor Interface

- Customizing Global Properties, Toolbar, and Menu Bar in the Contributor Interface

Customizing Publishing and Workflow

- Customizing Asset Forms for the Contributor Interface

- Customizing Workflow

- Working with RealTime Publishing Customization Hooks

- Understanding Asset and Publish Events in WebCenter Sites

- Customizing Content Audit Reports

Deploying Customizations: Adding Customizations to WebCenter Sites

67

Customizing the Tree in the Admin Interface

Administrative and editorial users of WebCenter Sites interact with various trees that display in the Admin interface. You can customize the Admin interface by modifying these trees.

Topics:

- About the Tree in the Admin Interface

- About Trees and Security

- About Tree Error Logging

67.1 About the Tree in the Admin Interface

The tree displays a number of nodes in the left pane of the Admin interface.

Figure 67-1   Admin Interface

WebCenter Sites tree tabs are created by the tree applet. You can create or modify your own trees by setting various parameters that are passed to the tree applet. The tree applet accepts several kinds of parameters:

- Applet-wide parameters, which control the overall appearance and behavior of the applet.

- Tree-specific parameters, which control the appearance and behavior of the tree.

- Node parameters, which control the appearance and behavior of individual nodes on the tree.

- OpURL Node parameters, which allow the tree to communicate with WebCenter Sites.

A set of tree tab tables in the database stores information about tree configuration, including tab names, what roles have access to a tab, and the path to the element that populates the tree tab with data. You enter information into these tables through the Tree Tabs screens. From the Admin interface, select the **Admin** tab and then click the **Tree** node.

67.1.1 Loading the Tree Tabs

For most of the default tree tabs supplied with WebCenter Sites, requests for tree data pass through the `OpenMarket/Gator/UIFramework/LoadTab` element. The `LoadTab` element performs several basic tasks, such as checking for session timeout.

For example, the **Product** tab completes the following steps as it loads:

1. Java code in the **Product** tab calls the `LoadTab` element.

2. The `LoadTab` element queries the TreeTab database tables to retrieve the elements that load the data for the Product tree's top-level nodes. In this case, the elements are the `OpenMarket/Xcelerate/ProductGroups/LoadTree` element and the `OpenMarket/Xcelerate/Product/LoadTree` element.

3. The `OpenMarket/Xcelerate/ProductGroups/LoadTree` element and the `OpenMarket/Xcelerate/Product/LoadTree` element query the database for assets that correspond to the tree nodes and stream back node data to the tree applet.

4. The tree applet parses the node data and opens the nodes.

5. Java code in the **Product** tab calls an element to initialize its global context menu, the `OpenMarket/Gator/UIFramework/LoadGlobalPopup` element. This element sends a `GetTypes` command to each tree loading element called by the **Product** tab. When the tree loading elements receive this command, they return a list of asset types whose start menu items should appear in the global context menu.

6. The `OpenMarket/Gator/UIFramework/LoadGlobalPopup` element finds the start menu items for the specified asset types and streams that information back to the tree.

Note that each asset type in the system must have a `LoadTree` element. The `LoadTree` element is a pointer to another element that actually loads the tree. If an asset type can have children, each of those children must have a `LoadTree` element. `LoadTree` elements have the following path:

`OpenMarket/Xcelrate/AssetType/MyAssetType/LoadTree`

where `MyAssetType` is the name of the asset type to which the `LoadTree` element refers.

`LoadTree` elements are called based on the asset type set in the **Section** field of the Manage Tree form.

The following table contains a list of several elements used by core asset types to load their trees.

Table 67-1   Asset Type Elements

Asset Type	Location	Description
Flex Groups	`OpenMarket/Gator/UIFramework/ LoadGroupNodes`	Displays a FlexGroup parent hierarchy and FlexAsset children.

Table 67-1   (Cont.) Asset Type Elements

Asset Type	Location	Description
Flex Assets	`OpenMarket/Gator/UIFramework/LoadOrphanNodes`	Displays flex assets that do not belong to a flex group.
Site Navigation Tree	`OpenMarket/Xcelerate/AssetType/Page/LoadSiteTree`	Displays the SitePlan tree.
Site Navigation Associations	`OpenMarket/Gator/UIFramework/LoadChildren`	Displays asset associations in the SitePlan tree.
Bookmarks	`OpenMarket/Gator/UIFramework/LoadActiveList`	Displays the Bookmarks tree.
Administrative Tree	`OpenMarket/Gator/UIFramework/LoadAdminTree`	Displays the Administrative tree.
Administrative Tree Helper Elements	`OpenMarket/Gator/UIFramework/Admin`	Loads helper elements for the Administrative tree.
Asset Types	`OpenMarket/Gator/UIFramework/LoadAdministrationAsset`	Displays an asset type node at the top level of the tree and the names of all assets of that type on lower levels of the tree.

To change the appearance or behavior of nodes in your tree, create a new tree loading element based on one of these standard elements. Your website administrator can then specify the element's name and the path to that element in the **Section Name** and **Element Name** fields of the New Tree form, located off the Tree Tabs form.

See About Creating Tree Tabs in *Oracle Fusion Middleware Administering Oracle WebCenter Sites*.

Note:

In some cases with a default JRE installation, some items in the tree may display with boxes in the name rather than UTF-8 characters. There are two ways to solve this font issue:

- Copy the supported font files to `$JRE_HOME/lib/fonts/fallback`

- Configure local fonts to physical fonts via mapping through one of the font configuration property files, as described here: `http://docs.oracle.com/javase/8/docs/technotes/guides/intl/fontconfig.html`.

For more information about modifying tree nodes, see Node Parameters.

67.1.1.1 Applet-Wide Parameters

Applet-wide parameters are set in the `TreeAppletParams.xml` element. To modify the tree applet's behavior, change the parameter values as shown in the following table.

Table 67-2   Applet-Wide Parameters

Parameter	Description
Debug	Turns debugging on and off. Valid values are `true` and `false`. If `Debug` is set to `true`, Java console debug and error messaging is turned on.
ServerBaseURL	Sets the base string to which all the node data URL strings are appended. For example, if the `ServerBaseURL` is set to `file://localhost`, and the value of the `LoadURL` parameter is `NodeReader.test`, then the URL used for loading the tree's child nodes is as follows: `file://localhost/NodeReader.test`
BackgroundColor	Sets the background color of the tree using a decimal RGB value. If this parameter is not set, the background color defaults to the color of the HTML frame in which the tree is embedded.
TotalPanes	Sets the number of tree tabs that are displayed. This value is set automatically.
URLTarget	The target frame in which to display node links. The default value is `XcelAction` (name of the pane on the right side of the browser window).

67.1.1.2 Tree-Specific Parameters

Tree-specific parameters are set by the Add New Tree Tab form and the `OpenMarket\Gator\UIFramework\TreeTabAdd.xml` element that creates the Add New Tree Tab form. To modify the tree's appearance or behavior, change the parameter values by using the form or by altering the `TreeTabAdd` element.

Table 67-3   Tree-Specific Parameters

Parameter	Description
Title	Sets the text that is displayed on the tab. This value is set in the **Title** field of the Manage Tree form, found on the **Admin** tab.
ToolTip	Sets the text that is displayed when the mouse pointer hovers over the tab index. This value is set in the **Tool Tip** field of the Manage Tree form, found on the **Admin** tab.
LoadURI	The URI of the page to call to retrieve a node's children. This value is set in the `TreeTabAdd` element.

Table 67-3   (Cont.) Tree-Specific Parameters

Parameter	Description
ActionURL	The URL of the page that performs a context menu action for a node in the tree. The default value points to the `OpURL.xml` element. This value is set in the `TreeTabAdd` element.
OpenIcon	The path to the icon to use when depicting an expanded node. The default is a plus sign (+). This value is set in the `TreeTabAdd` element.
CloseIcon	The path to the icon to use when depicting an unexpanded node. The default is a minus sign (–). This value is set in the `TreeTabAdd` element.
LineStyle	Sets whether lines connect the nodes of the tree. Valid values are `Angled` and blank; `Angled` is the default. If the parameter is set to `Angled`, lines connect the nodes. If the value is left blank, no lines connect the nodes. This value is set in the `TreeTabAdd` element.
RootID	Sets the ID of the root node. This string is used for specifying the node path. It defaults to the value of the `Title` parameter. This value is set in the `TreeTabAdd` element.
GlobalItems	This value is set in the **GlobalItems** field of the Manage Tree form, found on the **Admin** tab.
NodeItems	This value is set in the **NodeItems** field of the Manage Tree form, found on the **Admin** tab.

67.1.1.3 Node Parameters

The node parameters determine the appearance and behavior of the nodes in your tree. To define the appearance and behavior of these nodes, you write an element that sets the node parameters and passes their values to the `BuildTreeNode.xml` element, which creates the tree nodes.

Table 67-4   Node Parameters

Parameter	Description
Label	Specifies the text to be displayed for this node. The value does not have to be unique. The default value is "".
ID	A string identifier that is unique within the tree, used by WebCenter Sites to express selection paths. The ID is specified by WebCenter Sites.
ExecuteURL	The URI value of the page to be displayed when completing the `Execute` action. This value has the value of `ServerBaseURL` prepended to it. If the node is not executable, do not include this parameter in the node data.

Table 67-4   (Cont.) Node Parameters

Parameter	Description
URLTarget	The frame target for `ExecuteURL`. If `ExecuteURL` is not included in the node data, it defaults to the target specified in the Applet-wide parameters.
Description	An alternative to the string specified in `Label`, if you choose this option on the tree-wide context menu. The default value is "".
Level	The relative level of this node, represented by a number >= 0. A value of 0 indicates that the node is an immediate child of the node requesting the data. To load multiple levels of nodes at a time, set this value to a number greater than zero. The default value is 0.
Image	The URI for the image to be prepended to the label. If this field is not included in the node data, then no image is displayed for that node.

LoadURL	The URI for the subtree hierarchy. If this field is not included in the node data, this node requires no additional loading.
	
	The URL specified in this parameter must contain enough information so that the tree applet can find that node's children. For example, if your hierarchy is as follows:
	
	Product Tab / Reebok / Running Shoes
	
	the value of `LoadURL` is as follows:
	
	`ContentServer?pagename=OpenMarket/Gator/UIFramework/`
	`LoadTab&AssetType=ProductGroups&populate=OpenMarket/`
	`Xcelerate/AssetType/ProductGroups/LoadTree&op=`
	`load&parent=Variables.parentid`
	
	where `parentid` is the assetid of the "Running Shoes" asset, and `op` and `populate` are used by `LoadTab` to route to your tree load element.

OKAction	An action that is displayed in the node's context menu. This string may appear multiple times in the same node data set.
OpURL	The URL to execute a given action on the server. This value is prepended with the value of the `ServerBaseURL` parameter. Include this parameter in the node data unless the value of the `NodeItems` parameter is a null string, and thus has no `OKAction` specified.
RefreshKeys	Creates a key or set of set of keys which can be used to refresh the tree. Set the value to the ID of the current node.

The following excerpt from the `LoadAdministrationAsset` element sets the values of the node parameters and passes those values to the `BuildTreeNode` element.

The `ListofAsset` list referred to in this excerpt is a list of information about assets of a given type. This list was generated by a SQL query that is executed elsewhere in the element.

```
<CALLELEMENT NAME="OpenMarket/Gator/UIFramework/BuildTreeNode">
  <ARGUMENT NAME="Label"
    VALUE="ListofAsset.name"/>
```

```
            <ARGUMENT NAME="Description"
              VALUE="ListofAsset.description"/>
            <ARGUMENT NAME="ID"
              VALUE="Variables.TreeNodeID"/>
            <ARGUMENT NAME="OpURL"
              VALUE="ContentServer?pagename=
              OpenMarket/Gator/UIFramework/TreeOpURL&#38;
              AssetType=Variables.AssetType"/>

            <ARGUMENT NAME="ExecuteURL"
              VALUE="ContentServer?pagename=
              OpenMarket/Gator/UIFramework/TreeOpURL&#38;
              AssetType=Variables.AssetType&#38;n0_=
              Variables.packedTreeNodeID&#38;op=displayNode"/>

            <ARGUMENT NAME="OKActions"
              VALUE="Status;Inspect;Edit;Delete;refresh"/>
            <ARGUMENT NAME="Image"
              VALUE="Xcelerate/OMTree/TreeImages/AssetTypes/Variables.AssetType.gif"/>
            <ARGUMENT NAME="RefreshKeys"
              VALUE="ListofAsset.id"/>
        </CALLELEMENT>
```

To customize the appearance or behavior of tree nodes, copy one of the standard elements and modify the node arguments. Note that tree loading elements are passed the following variables, so any tree loading element that you create or customize must take these variables into account.

Variables Passed in by the LoadTree Element:

- `AssetType`: Set to the section name that was created using the New Tree form

- `op`: Set to `init`

Variables Passed in by the LoadGlobalPopup Element:

- `command`: Set to `GetTypes`

- `AssetType`: Set to the section name that was created using the New Tree form

- `varname`: You set this with a comma-separated list of asset types for which you want to display start menu items

- `popupvar`: You set to either `true`, to add items to the global context menu, or `false`, if you do not have to add items to the context menu

67.1.1.4 Adding a Command Node Context Menu

Each node on the tree has a menu that displays when the user right-clicks the mouse. Commands on this menu allow you to refresh the node or load pages in the right side of the browser window. You can add commands to a node context menu that allow you to load forms such as the status and publish forms. Any form that can be called using an asset type and ID is a good candidate for being called by a node context menu command.

Add a command to the node context menu by completing the following steps:

1. Add the new command, exactly as you want it to appear, into the node's **OKActions** field.

2. In the element referred to in the node's `OpURL` (usually the `TreeOpURL` element), add a new `IF` statement that calls the form you want to load.

For example, the following code from the `TreeOpURL` element opens a node:

```
<IF COND="Variables.op=displayNode">
    <THEN>
     <callelement NAME="OpenMarket/Gator/UIFramework/TreeIDFromPath">
     <argument NAME="TreePath" VALUE="Variables.TreeNodePath"/>
     </callelement>
        <setvar NAME="id" VALUE="Variables.ID"/>
        <callelement NAME="OpenMarket/Xcelerate/UIFramework/
ApplicationPage">
        <argument NAME="ThisPage" VALUE="ContentDetailsFront"/>
        <argument NAME="contentfunctions" VALUE="true"/>
        <argument NAME="AssetType" VALUE="Variables.AssetType"/>
     </callelement>
    </THEN>
```

67.1.2 Refreshing the Tree

Elements that can alter the tree are responsible for refreshing the tree so that it displays current data. There are three different types of refresh actions that you can specify:

- Self: Refreshes the children of the specified node

- Parent: Refreshes the specified node and its children

- Root: Refreshes the entire tree

There are two steps to refreshing the tree:

1. Code your tree customization elements so that the tree nodes that you want to refresh have `RefreshKeys`. `RefreshKeys` are usually the asset ID of the current node, and allow the refresh to take place.

2. Call the `OpenMarket/Xcelerate/UIFramework/UpdateTreeOMTree` element, and pass the element the `_TreeRefreshKeys_` variable, specifying the type of refresh you want in the variable value.

You set the `RefreshKeys` for a node by passing the `RefreshKeys` argument to the `BuildTreeNode` element, as shown in the code sample in Node Parameters.

To refresh the tree, call the `OpenMarket/Xcelerate/UIFramework/UpdateTreeOMTree` element, as shown in the following example:

```
<CALLELEMENT NAME="OpenMarket/Xcelerate/UIFramework/UpdateTreeOMTree">
    <ARGUMENT NAME= "_TreeRefreshKeys_" VALUE= "Root:ActiveList"/>
</CALLELEMENT>
```

67.2 About Trees and Security

WebCenter Sites uses security roles to control access to the tree in the WebCenter Sites Admin interface. You need to assign the `xceladmin` ACL to the users of the system-defined nodes such as **Admin** and **Workflow**.

Additional control is available by setting properties in `wcs_properties.json`. For example, `xcelerate.showSiteTree` determines whether the tree is displayed by default; `xcelerate.restrictSiteTree` determines which users can display or

hide the tree. See Managing Users and Security in *Oracle Fusion Middleware Administering Oracle WebCenter Sites*.

67.3 About Tree Error Logging

All tree-related error and debug messages are logged to the Java Console. You can turn debugging on and off by supplying a value for the `Debug` parameter when you create a tree.

Note that enabling debug affects performance, so error logging in the delivery system should generally be turned off.

68

About Customizing Components of the Contributor Interface

To help improve content contributors' and marketers' productivity and save their time, you can customize Contributor interface's components such as dashboard, search views, and asset forms.

Topics:

- Before You Begin
- What Can You Customize in the Contributor Interface?
- Where to Find Sample Code?
- Where to Begin?

> **Note:**
>
> The Contributor interface is designed to be used by content providers, rather than developers. Therefore, the following system-defined asset types can be displayed (inspected) in the Contributor interface, but they cannot be created (or edited) in the Contributor interface: Template, CSElement, SiteEntry, DimensionSet, Dimension, Attribute Editor, Parent Definitions, Attributes, and Flex Definitions.
>
> Similarly, the following system-defined asset types are not accessible from the Contributor interface: `FW_Application`, and `FW_View`. Assets of these types can be created (edited) and displayed only in the Admin interface.

68.1 Before You Begin

To be able to customize the Contributor, you should have a working knowledge of the interface, experience with Java, JavaScript, and HTML, and solid familiarity with WebCenter Sites development tools.

See Exploring the Contributor Interface in *Oracle Fusion Middleware Using Oracle WebCenter Sites* and Configuring the User Interfaces in *Oracle Fusion Middleware Administering Oracle WebCenter Sites*.

68.2 What Can You Customize in the Contributor Interface?

The components you can customize are dashboard, search views, asset forms, configuration properties, toolbar, and menu bar.

- Dashboard. See Customizing the Contributor Interface Dashboard.
- Search views. See Customizing Search Views of the Contributor Interface.

- Global and site-specific configuration properties, toolbar, and menu bar. See Customizing Global Properties, Toolbar, and Menu Bar in the Contributor Interface.

- Asset forms. See Customizing Asset Forms for the Contributor Interface.

68.3 Where to Find Sample Code?

Here is some sample code that illustrates how you can customize the interface. Other code is either packaged in WebCenter Sites or available independently. Paths to such code are listed in respective topics.

68.4 Where to Begin?

The Contributor interface framework contains a component called the *UI Controller*, which handles most of the interface-related requests, except for those pertaining to asset forms.

The UI Controller is described in Understanding the Contributor Interface Framework and UI Controller.

- If you are customizing the dashboard, search views, configuration properties, toolbars, or menu bars, you should start with Understanding the Contributor Interface Framework and UI Controller to obtain basic information about the concepts and code for the customization process.

- If you are customizing asset forms, you can skip to Customizing Asset Forms for the Contributor Interface for information about modifying asset form headers and building an attribute editor.

69

Understanding the Contributor Interface Framework and UI Controller

In the Oracle WebCenter Sites: Contributor interface framework, specifically the UI Controller handles all interface requests and processes elements.

Topics:

- About the Contributor Interface Framework
- UI Controller
- Custom Elements

69.1 About the Contributor Interface Framework

The framework of the Contributor interface sits on top of the Services Layer and handles client requests.

This figure shows the framework, consisting of the Presentation Layer and UI Controller.

Figure 69-1   Contributor Interface Framework
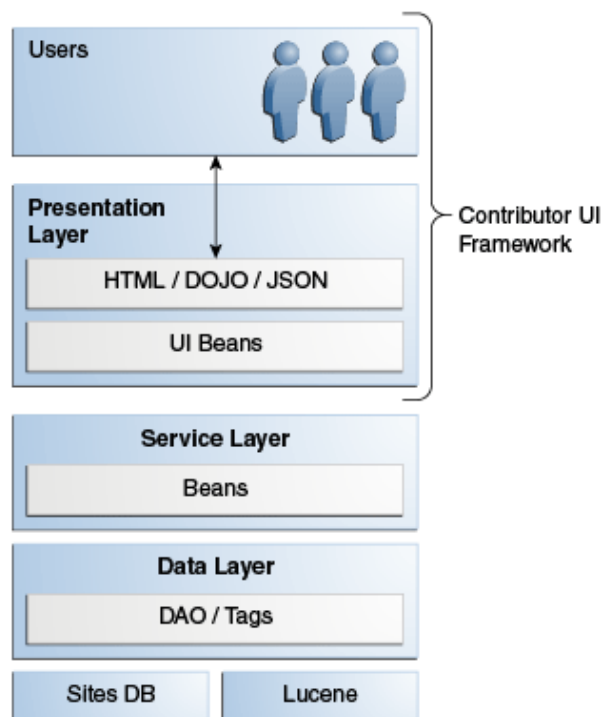
The Presentation Layer consists of elements that render views and elements that generate a response. The UI Controller processes the requests it receives from the Contributor interface, as explained in UI Controller.

> **Note:**
>
> The UI Controller is not used to process requests pertaining to asset forms, given that asset forms exist outside the Contributor framework. See Customizing Asset Forms for the Contributor Interface.

69.2 UI Controller

The UI Controller processes requests in configuration, action, and presentation phases. In each phase, it determines the corresponding element by a naming convention. There is a process by which the UI Controller checks for custom elements.

Topics:

- How the UI Controller Processes Requests

- UI Controller Processing an Element Request: Example

69.2.1 How the UI Controller Processes Requests

The UI Controller can be reached by invoking the `fatwire/ui/controller` SiteCatalog entry. The UI Controller requires the incoming request to provide at least one parameter, `elementName`, which determines the controller element to be executed. For example, the following URL invokes the controller element `Foo/Bar`:

```
http://localhost:7001/sites/ContentServer?pagename=fatwire/ui/
controller/controller&elementName=Foo/Bar
```

A controller element is processed in the following three phases:

1. Configuration phase

2. Action phase

3. Presentation phase

where each phase consists of running a distinct element. For each phase, the corresponding element name is determined by a naming convention, described below.

> **Note:**
>
> A controller element is any element that can be invoked through the UI Controller.
>
> In each of the phases, the UI Controller first tests for the custom element specific to that phase. The process flow is illustrated in the steps of UI Controller Processing an Element Request: Example.

1. Configuration Phase

This phase consists of evaluating the configuration element. The configuration element is meant to contain configuration settings used by the controller element being invoked. The expected element name is `<controllerElementName>Config`, where

`<controllerElementName>` is the value of the `elementName` parameter. For instance, in our example, where the controller element name is assumed to be `Foo/Bar`, the expected name of the configuration element is `Foo/BarConfig`.

The Configuration phase is based on Apache Commons Configuration and requires configuration data to be formatted as a valid XML document. For example:

```
<myconfig>
    <foo>123</foo>
    <bar>foobar</bar>
</myconfig>
```

The XML configuration data is evaluated into a configuration object, that is, an instance of `org.apache.commons.configuration.beanutils.ConfigurationDynaBean`, which is kept in the request scope, where it is identified by the name of the XML root element. In our example, the configuration object can be accessed in the Action phase or Presentation phase as follows:

```
ConfigurationDynaBean configBean =
(ConfigurationDynaBean)request.getAttribute("myconfig");
```

where `myconfig` matches the name of the top-level XML element in the configuration element. More information about Apache commons configuration can be obtained at the following URL: http://commons.apache.org/configuration.

> **Note:**
>
> About Configuration Elements in the Configuration Phase:
>
> 1. The Configuration phase is conditional. If the element `<controllerElementName>Config` does not exist, the UI Controller skips this phase and moves on to the next phase without creating a configuration object.
>
> 2. Unlike in the other two phases, the configuration element is not evaluated directly (using, for example, `ics.callElement`). Instead, it is invoked through the `fatwire/ui/controller/readConfiguration` SiteCatalog entry, using `ics.ReadPage()`, allowing to capture its output.
>
> 3. While creating the configuration object, the controller checks if there is a custom configuration available for the requested element under CustomElements. If there is one, the controller, by default, merges the custom configuration with the corresponding system configuration. This is the default behavior. If you do not want to merge the custom configuration with the system configuration, specify `merge=false` for the field **resdetails1** or **resdetails2** in the ElementCatalog entry for the custom configuration element.
>
>   For example, if the system configuration is:
>
>    ```
>    <myconfig>
>        <foo>123</foo>
>        <bar>foobar</bar>
>    </myconfig>
>    ```
>
>   And if the custom configuration is:
>
>    ```
>    <myconfig>
>        <foo1>456</foo1>
>        <bar1>foo1bar1<bar1>
>    </myconfig>
>    ```
>
>   Then the controller merges the system configuration and custom configuration to have the following:
>
>    ```
>    <myconfig>
>        <foo>123</foo>
>        <bar>foobar</bar>
>        <foo1>456</foo1>
>        <bar1>foo1bar1</bar1>
>    </myconfig>
>    ```

2. Action Phase

In this phase, the UI Controller evaluates the action element. The expected name of the action element is `<controllerElementName>Action`. In our example, the action element name is `Foo/BarAction`.

The action element is meant to contain arbitrary business logic. It typically builds Java objects in the request scope, to be consumed by the next phase.

> **Note:**
>
> The Action phase is conditional. If the element
> `<controllerElementName>Action` does not exist, then the UI Controller
> skips this phase and moves on to the Presentation phase.

3. Presentation Phase

In this last phase, the UI Controller evaluates the presentation element, whose name depends on the content type of the generated output. The UI Controller can serve either HTML (the default behavior) or JSON. The element name would then be `<controllerElementName>Html` or `<controllerElementName>Json`.

In our example, the UI Controller attempts to evaluate `Foo/BarHtml`, because HTML is the default content type. To generate JSON data instead, you must explicitly specify a response type as follows:

```
http://localhost:7001/sites/ContentServer?pagename=fatwire/ui/
controller/controller&elementName=Foo/Bar&responseType=json
```

In this case, the UI Controller attempts to evaluate the presentation element called `Foo/BarJson`.

69.2.2 UI Controller Processing an Element Request: Example

When the UI Controller processes any element request, it tests for the custom element as follows:

In each phase (Configuration, Action, and Presentation), the UI Controller first looks for the custom element specific to that phase. If the custom element is not found, the UI Controller looks for the default element. If the default element is not found, the UI Controller skips the phase and moves on to the next phase.

The steps below explain, by example, how the UI Controller processes an element request. In this example, the request is for an existing element named `UI/Layout/LeftNavigation`, and the response type is `Html`:

1. **Configuration Phase.** The UI Controller looks for the `LeftNavigation` element's configuration. That is, the UI Controller looks for the element named `LeftNavigationConfig.jsp` under `CustomElements` (in the `ElementCatalog`). If the element exists, the UI Controller reads this element. Otherwise, it reads the default element `LeftNavigationConfig.jsp` (in `UI/Layout/`). The UI Controller then generates the configuration object and keeps this object in the request scope.

   An alternative is to pass the configuration file name as an argument to the UI Controller call. The passed parameter is named `configName`. If `configName` is passed, the UI Controller looks for the element specified in that parameter.

2. **Action Phase.** The UI Controller now looks for the element `LeftNavigationAction.jsp`. If it finds the element under `CustomElements`, the UI Controller executes this element. Otherwise, it executes the default `LeftNavigationAction.jsp` element (in `UI/Layout/`).

3. **Presentation Phase.** In the current example, the response type is Html. Therefore, the UI Controller looks for the element `LeftNavigationHtml.jsp`. If it finds the element under `CustomElements`, the UI Controller executes this element to

generate an `Html` response. Otherwise, it executes the default
`LeftNavigationHtml.jsp` element (in `UI/Layout/`).

69.3 Custom Elements

When you are customizing the Contributor interface, remember to store your custom
elements in the recommended location. Make sure that you know how the custom
elements are located by the UI Controller.

Topics:

- Element Storage

- How the UI Controller Locates Elements

- Element Naming Conventions

69.3.1 Element Storage

The framework of the Contributor interface allows developers to keep their custom
elements separate from the system default elements, in accordance with best practices.

> **Note:**
>
> Oracle recommends not modifying the system's default configurations.
> Instead, create your own custom elements and store them under
> `CustomElements` of the ElementCatalog to ensure their preservation during
> upgrades.

The path to a custom element depends on whether the element is global, site-specific,
site- and asset type- specific, or just asset type-specific. This figure shows paths to
custom elements.

Figure 69-2   Paths to Custom Elements
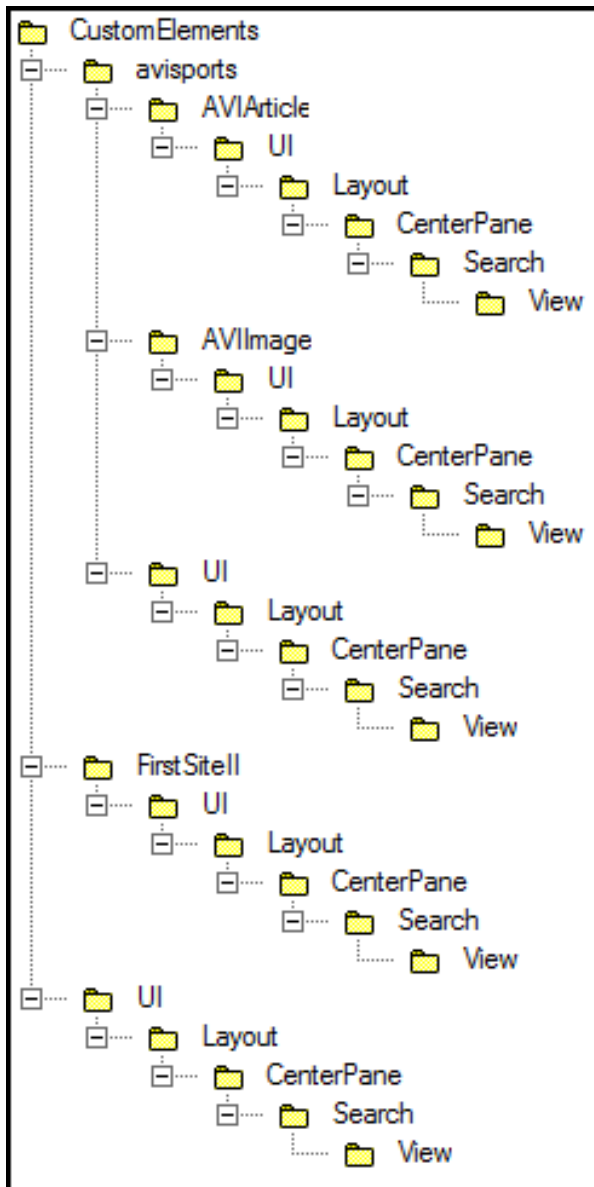
69.3.2 How the UI Controller Locates Elements

When the UI Controller looks for an element:

1. The UI Controller first looks for the customized version of the element by traversing all paths under `CustomElements` in the following order:

    a. Site-specific and asset type-specific paths

    b. Asset type-specific paths

    c. Site-specific paths

    d. Global paths

    (For an example of paths, see Figure 69-2.)

2. If the custom element is not found, the UI Controller uses the system-defined element.

> **Note:**
>
> For the UI Controller to use the asset type-specific element, the `assetTypeParam` parameter must be passed with a valid asset type as its value.

69.3.3 Element Naming Conventions

When referring to a system-defined element or sample element packaged with WebCenter Sites, this guide provides the full path to the element. The full path always begins with `UI/Layout/`. For example, the system-defined element `DashBoardContentsConfig.jsp` is presented as follows in this guide:

`UI/Layout/CenterPane/DashBoardContentsConfig`

When referring to a custom-defined element that you create, this guide provides only the name of the element (JSP), given that its path is unknown. For example:

`DashBoardContentsConfig.jsp`

It is assumed that the custom element is stored under `CustomElements`.

70

Customizing the Contributor Interface Dashboard

Before you start customizing the dashboard, familiarize yourself with its configuration. You can also use a sample code to try out dashboard customization.

Topics:

- About Dashboard Customization
- Customizing the Dashboard
- Examples of Customizing the Dashboard

70.1 About Dashboard Customization

When you log in to the Contributor interface, the dashboard is displayed. By default, the dashboard opens the following ready-to-use widgets: Bookmarks, SmartLists, Checkouts, and Assignments.

Figure 70-1   Dashboard with Default Widgets

You can customize the following portions of the dashboard and its widgets:

- Number of columns

- Column width

- Display name, height, and dashboard position

- Number of widgets

70.2 Customizing the Dashboard

When you're customizing the dashboard, you can override the controller element which generates the system-defined dashboard. You can configure a dashboard with a global setting or make it site-specific. On the dashboard, you can customize default widgets, add new widgets, and delete those that content and marketing teams no longer need.

- The system-defined dashboard is generated by the controller element `UI/Layout/CenterPane/DashboardContentsConfig`.

  To customize the dashboard, override this element by creating your own `DashBoardContentsConfig.jsp` under `CustomElements` and customizing its properties.

  > **Note:**
  >
  > When a new widget is added or an existing widget is updated, you must clear user preferences in the WEM UI for the changes to take place.

The `UI/Layout/CenterPane/DashBoardContentsConfig` element is shown next, followed by property descriptions in the next table.

Element UI/Layout/CenterPane/DashboardContentsConfig

```
<dashboardconfig>
    <dashboardlayout>
        <numberofcolumns></numberofcolumns>
        <columnwidths></columnwidths>
    </dashboardlayout>
    <components>
        <component id="widgetId">
            <name>widgetName</name>
            <url>widgetURL</url>
            <height>height_in_px</height>
            <dragRestriction>true | false </dragRestriction>
            <column>number_of_column_in_which_to_display_widget</column>
        </component>
            …
            …
            …
    </components>
</dashboardconfig>
```

Table 70-1   Properties in `UI/Layout/CenterPane/DashBoardContentsConfig.jsp`

Property	Description	Value
`<numberofcolumns>`	Number of columns in the dashboard display.	Integer greater than 0. The system default is 2.
`<columnwidths>`	Comma-separated widths of columns.	For example, if there are 3 columns in `<numberofcolumns>` then the `<columnwidths>` can be `30,30,40`.
`<components>`	This section is used to define dashboard widgets.	N/A
`<component>`	Used to define a single widget.	N/A
`<id>`	ID of the widget.	Alpha-numeric value unique across widgets. Special characters are not allowed.
`<name>`	Displayed name of the widget.	Arbitrary string.
`<url>`	Controller URL.	The file location of the widget in the `UI/Layout/CenterPane/DashBoard/<Your_Element>/` directory.
`<height>`	Height of the widget.	Height in pixels. For example, `300px`.
`<dragRestriction>`	Restricts dragging of the widget.	`true	false`
`<column>`	The column in which the widget is displayed.	`1` to `n`, where n is the value specified in `<numberofcolumns>`.

70.3 Examples of Customizing the Dashboard

You might want to ask content and marketing teams which widgets can help them be more productive. You can add those widgets to the Contributor dashboard.

To add a new widget:

1. Create the widget element.

2. Register the new widget in your custom `DashBoardContentsConfig.jsp` element.

70.3.1 Adding a Hello World Widget

These steps show how to create and register the simple widget shown in this figure.

Figure 70-2   Hello World Widget

To add your widget to the dashboard create your widget as follows:

1. Create a JSP element under CustomElements. In this example, we name the element `HelloWorldHtml`.

2. For widget code, you can navigate to the sample file provided with this guide and copy its content.

To register your widget (add it to the dashboard):

1. Open your custom `DashBoardContentsConfig.jsp`, locate the `<components>` section, and add the newly created widget's specifications. For example:

```
<component id="helloworld">
  <name>Hello World</name>
  <url>Path_to_your_widget_under_CustomElements</url>
  <height>300px</height>
  <closable>false</closable>
  <open>true</open>
  <dragRestriction>true</dragRestriction>
  <style>checkoutPortlet</style>
  <column>2</column>
</component>
```

2. Go to the `<applicationServer_install_directory>/webapps/`
    `<cs_context>/WEB-INF/classes/ReqAuthConfig.xml` file and add the
    path to the sample element, under the `excludedControllerElements` list. In
    our example, the path is:

```
<property name="excludedControllerElements">
   <list>
       <value>UI/Layout/CenterPane/DashBoard/HelloWorld</value>
```

```
            </list>
        </property>
```

3. Refresh the home page of your Contributor interface. The new widget is displayed on your dashboard.

70.3.2 Adding a Widget that Shows Recently Modified Assets

In this section, you create a widget that shows which assets were modified in the past week. After completing the steps in this section, your dashboard displays a widget similar to the one in this figure.

Figure 70-3   Recently Modified Assets Widget
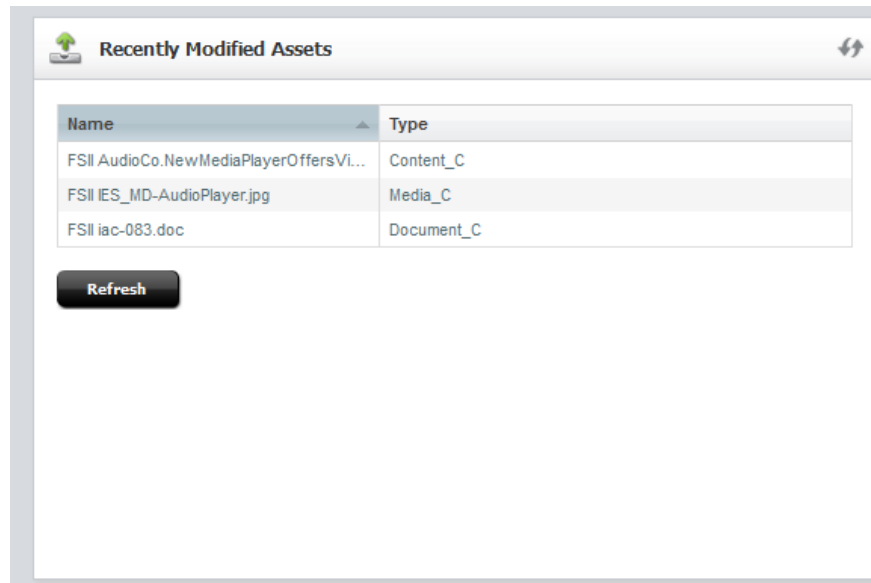
To add your widget to the dashboard, create your widget as follows:

1. Create an `Action` JSP element under `CustomElements`. In this example, we name the element `RecentlyModifiedAssetsAction.jsp`. For the widget code, you can navigate to the sample file provided with this guide and copy its content.

2. Create a `Json` JSP element for the Action element created in the previous step. In this example, we name the element `RecentlyModifiedAssetsJson.jsp`. For the code, you can navigate to the sample file provided with this guide and copy its content. Place the element in the same location as the `RecentlyModifiedAssetsAction.jsp` element.

3. Create a presentation element under CustomElements for your widget. Name the element after the widget element. In this example, we name the display element `RecentlyModifiedAssetsHtml.jsp`. For the code, you can navigate to the sample file provided with this guide and copy its content.

Note:

The presentation element calls the `RecentlyModifiedAssetsAction.jsp` element. Enter the path to that element.

To register your widget (add it to the dashboard):

1. Open your custom `DashBoardContentsConfig.jsp`, locate the `<components>` section, and add the newly created widget's specifications. For example:

```
<component id="myrecent">
<!-- a unique identifier for the component. This must be unique among all the
components. It can be alpha numeric but no special characters allowed -->
      <name>Recently Modified Assets</name>
      <url>Path_to_your_custom_widget's presentation_element</url>
      <height>300px</height>
      <closable>false</closable>
      <open>true</open>
      <dragRestriction>false</dragRestriction>
      <style>checkoutPortlet</style>
      <column>2</column>
</component>
```

2. Go to the `<applicationServer_install_directory>/webapps/` `<cs_context>/WEB-INF/classes/ReqAuthConfig.xml` file and add the path to the sample element, under the `excludedControllerElements` list. In our example, the path is:

```
<property name="excludedControllerElements">
   <list>
      <value>UI/Layout/CenterPane/DashBoard/RecentlyModifiedAssets</value>
   </list>
</property>
```

3. Refresh the dashboard to see the newly configured widget.

71

Customizing Search Views of the Contributor Interface

Different types of search views are available in the Contributor interface. Become familiar with these views and their configuration elements.

Topics:

- About Search View Customization
- Customization Processes
- Customizing Undocked Views
- About Customizing Docked Views
- Customizing Sort Menus and Tooltips

71.1 About Search View Customization

When users log in to the Contributor interface and access their sites, they can perform a simple or advanced search to locate the required assets. Search results are then presented in either List view or Thumbnail view. You can customize some features of different search views.

See Finding and Organizing Assets and Working with the Search Results List in *Oracle Fusion Middleware Using Oracle WebCenter Sites*.

Topics:

- Types of Search Views
- What You Can Customize in Search Views
- View-Rendering Process
- Configuration Elements for Search Views

71.1.1 Types of Search Views

The search results panel can be either undocked or docked and displayed as a List view or Thumbnail view. Thus, the Contributor interface includes the following views:

- List Undocked
- List Docked
- Thumbnail Undocked
- Thumbnail Docked

An undocked view opens only when no assets are open for editing. A docked view is attached to assets in edit mode and therefore opens only when an asset is open in edit mode.

71.1.2 What You Can Customize in Search Views

The following figure summarizes the features you can customize in List view. See Customizing Undocked Views, which also applies to docked views.

Sort menus and tooltips are customized separately. See Customizing Sort Menus and Tooltips.

Which view opens by default for a given mode depends on your configuration settings and the user's search habits. For example, if you set Thumbnail view as the default view for undocked mode, then Thumbnail view opens when the user first runs search in undocked mode and continues to open until the user switches to List view. (Search remembers the user's choice until browser cookies are cleared.)

The following figure shows the customizable features in List view.

Figure 71-1   Customizable Features in List View
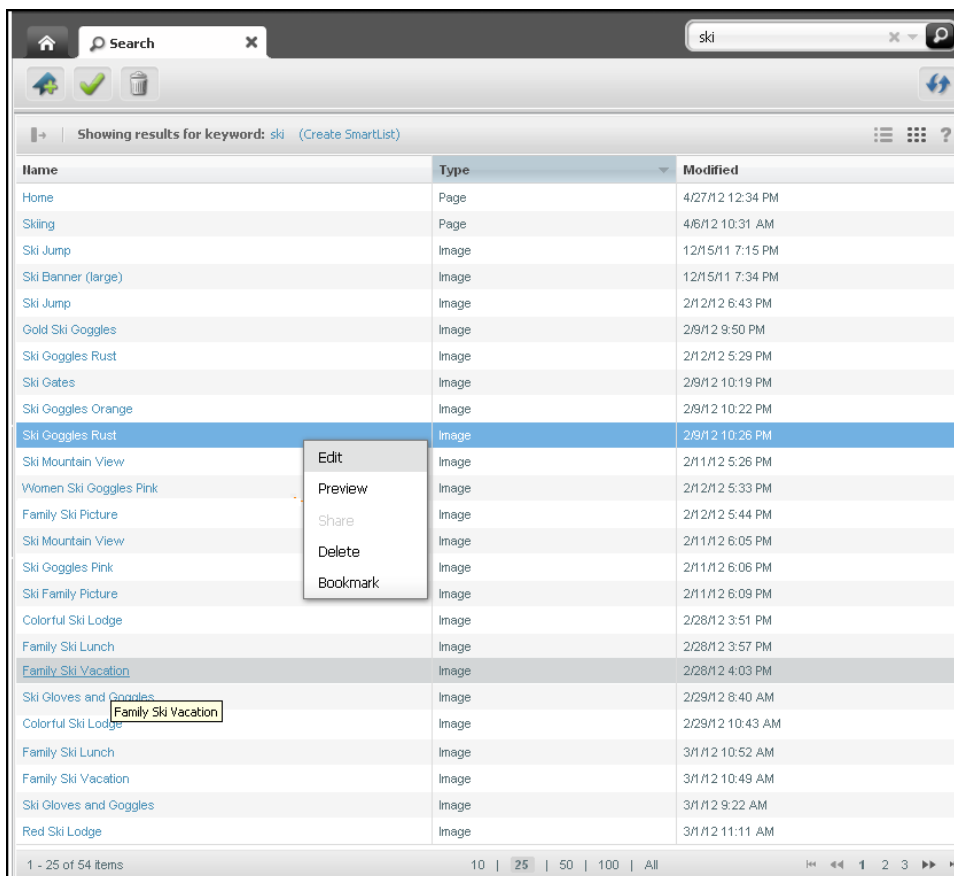
Customizable features for List view include:

- Maximum number of items to return

- Number of rows per page

- Fields (columns) to display

- Column display name

- Column width

- Format of date and other fields

- Default sort field and sort order

- Sort menu (docked mode)

- Context (right-click) menu

- Tooltip (docked mode)

The following figure summarizes the features you can customize in Thumbnail view.

Figure 71-2   Customizable Features in Thumbnail View
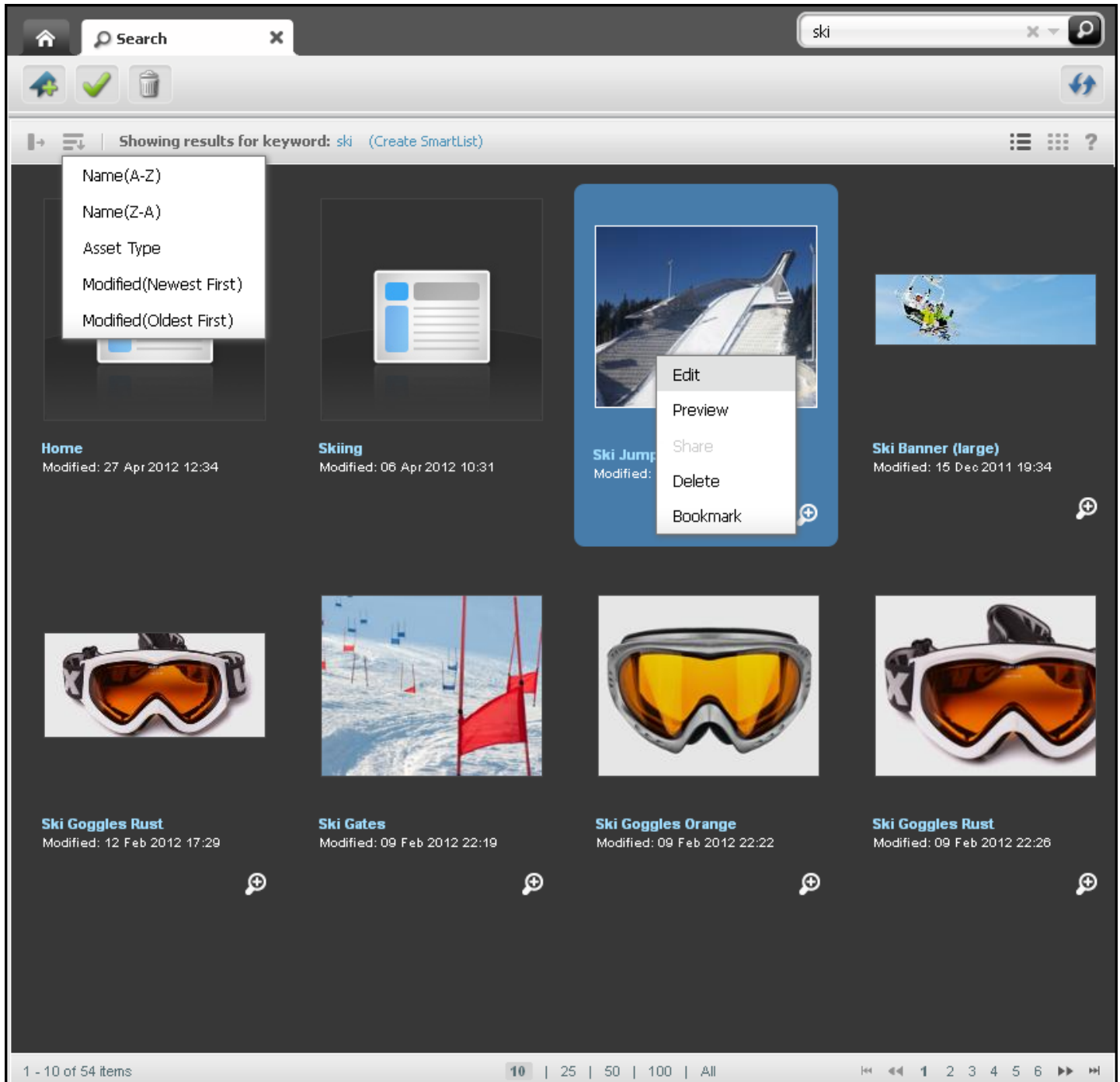
Customizable features for Thumbnail view include:

- Maximum number of items to return

- Number of rows per page

- Asset types for which special thumbnails are shown

- Fields to display

- Format of date and other fields

- Default sort field and sort order

- Sort menu

- Context (right-click) menu

- Tooltip (docked mode)

71.1.3 View-Rendering Process

System-defined and custom-defined views are rendered by similar processes. To illustrate, we begin with system-defined views.

System-defined views are rendered by the following elements (JSPs), whose names for undocked and docked views differ only by the `Docked` prefix.

When undocked:

- List view is rendered by the element: `UI/Layout/CenterPane/Search/View/ListViewHtml`

- Thumbnail view is rendered by the element: `UI/Layout/CenterPane/Search/View/ThumbnailViewHtml`

When docked:

- List view is rendered by the element: `UI/Layout/CenterPane/Search/View/DockedListViewHtml`

- Thumbnail view is rendered by the element: `UI/Layout/CenterPane/Search/View/DockedThumbnailViewHtml`

Rendering of undocked and docked views is similar (except that the names of elements for docked views start with `Docked`). The following steps illustrate the rendering of undocked views.

1. When a user runs a search routine, the search functionality determines the user's current view, which is either the default view or a subsequently chosen view.

   > **Note:**
   >
   > "Default view" is the view that the system renders the first time search is run. (List view is the system-defined default view for both undocked and docked modes.) If the user switches to a different view, search remembers and continues to display the user's choice until browser cookies are cleared.

2. Search functionality reads `UI/Layout/CenterPane/Search/SearchResultsConfig` to obtain the path to the element that initiates the rendering of the view:

   - If the user is running search for the first time, or continues using the default view, search reads the value of the `<defaultview>` property.

   - If the user's view is other than the default view, search reads the value of either the `<listview>` or `<thumbnailview>` property (depending on which view was determined in step 1).

3. If search determines that List view must be rendered, it reads the element `UI/Layout/CenterPane/Search/View/ListViewConfig` and invokes `UI/Layout/CenterPane/Search/View/ListViewHtml`, which then renders the list view. If search determines that the Thumbnail view must be rendered, it reads the element `UI/Layout/CenterPane/Search/View/ThumbnailViewConfig` and invokes `UI/Layout/CenterPane/Search/View/ThumbnailViewHtml`, which then renders the Thumbnail view.

You can override all of the above system-defined elements by customizing your own identically named elements and placing them under `CustomElements` to actualize the changes shown in Figure 71-1 and Figure 71-2. You can also customize individual features, such as sort menus and tooltips, by using the elements `UI/Layout/CenterPane/Search/View/SearchTopBarConfig` and `UI/Layout/CenterPane/Search/View/SearchToolTipHtml` respectively.

For a comprehensive list of elements, see Configuration Elements for Search Views.

71.1.4 Configuration Elements for Search Views

This section summarizes the JSP elements you use to customize search views.

- **System-defined configuration elements**: You can configure identically named elements to customize search views and searches that are global or specific to a site, asset type(s), or site and asset type(s). All customized elements should be stored under `CustomElements` (for an example, see Figure 71-3). For a summary of the elements, see the following tables:

  - This table lists system-defined configuration elements that define ready-to-use undocked views (all of the element names end with `Config`).

Table 71-1   Configuration Elements for Undocked Search Views

Path to Configuration Element (JSP)	Description	See …
`UI/Layout/CenterPane/Search/SearchResultsConfig`	Element for setting the default search view (List view or Thumbnail view) in undocked mode.	Setting the Default Undocked View to List or Thumbnail
`UI/Layout/CenterPane/Search/View/ListViewConfig`	Element for configuring the undocked List view.	Customizing the Undocked List View
`UI/Layout/CenterPane/Search/View/ThumbnailViewConfig`	Element for configuring the undocked Thumbnail view.	Customizing the Undocked Thumbnail View

  - This table lists system-defined configuration elements that define ready-to-use docked views (all of the element names end with `Config`).

Table 71-2   Configuration Elements for Docked Search Views

Path to Configuration Element (JSP)	Description	See …
`UI/Layout/CenterPane/Search/DockedSearchResultsConfig`	Element for setting the default search view (List view or Thumbnail view) in docked mode.	Customization Processes
`UI/Layout/CenterPane/Search/View/DockedListViewConfig`	Element for configuring the docked List view.	Customization Processes

Table 71-2 (Cont.) Configuration Elements for Docked Search Views

Path to Configuration Element (JSP)	Description	See …
`UI/Layout/CenterPane/Search/` `View/DockedThumbnailViewConfig`	Element for configuring the docked Thumbnail view.	Customization Processes

– This table lists system-defined elements for customizing a search view's individual features, such as sort menus and tooltips (element names end with either `Config` or `Html`).

Table 71-3 Configuration and Presentation Elements for Other Features in Search Views

Path to Configuration Element (JSP)	Description	See …
`UI/Layout/CenterPane/Search/` `View/SearchTopBarConfig`	Element for configuring fields as sort options in the sort drop-down menus for docked List, undocked Thumbnail, and docked Thumbnail views.	Customizing Sort Menus
`UI/Layout/CenterPane/Search/` `View/SearchToolTipHtml`	Element for configuring tooltips for docked views (List and Thumbnail). This element enables you to configure tooltip appearance and custom messages.	Customizing Tooltips for Search Results
`UI/Config/GlobalHtml`	Element for configuring context (right-click) menus. This element is valid for all search views.	Customizing Context Menus

- **Custom elements**: This table lists sample custom elements that are packaged with WebCenter Sites to help illustrate customization code.

Table 71-4 Custom Sample Elements for Search Views

Path to Sample Element	Description
`CustomElements/avisports/AVIArticle/UI/` `Layout/CenterPane/Search/View/` `ThumbnailViewConfig`	Configuration element for undocked Thumbnail view for the AVIArticle asset type in the avisports sample site.
`CustomElements/avisports/AVIArticle/UI/` `Layout/CenterPane/Search/View/` `DockedThumbnailViewConfig`	Configuration element for docked Thumbnail view for AVIArticle asset type in avisports site.
`CustomElements/avisports/AVIImage/UI/` `Layout/CenterPane/Search/View/` `ThumbnailViewConfig`	Configuration element for undocked Thumbnail view for the AVIImage asset type in the avisports sample site.
`CustomElements/avisports/AVIImage/UI/` `Layout/CenterPane/Search/View/` `DockedThumbnailViewConfig`	Configuration element for docked Thumbnail view for the AVIImage asset type in the avisports sample site.

Table 71-4   (Cont.) Custom Sample Elements for Search Views

Path to Sample Element	Description

`CustomElements/avisports/UI/Layout/`
`CenterPane/Search/View/`
`ThumbnailViewConfig`
	Configuration element for undocked Thumbnail view for the avisports sample site.

`CustomElements/avisports/UI/Layout/`
`CenterPane/Search/View/`
`DockedThumbnailViewConfig`
	Configuration element for docked Thumbnail view for the avisports sample site.

71.2 Customization Processes

The steps you perform to customize views in undocked and docked mode are similar with only a few differences. Read further to know what those differences are.

- **Customizing undocked and docked views:**

  When customizing undocked views, follow instructions in Customizing Undocked Views and name your configuration elements (JSPs) as shown in that section (also in Table 71-1). When customizing docked views, also follow instructions in Customizing Undocked Views, but name your configuration elements as shown in Table 71-2 (that is, include the `Docked` prefix).

- **Customizing sort menus and tooltips for search views:**

  Elements for creating sort menus and tooltips apply to both undocked and docked mode. Name the elements exactly as shown in Table 71-3 (and Customizing Sort Menus and Tooltips, regardless of mode.

- **To display a field in docked List view or docked Thumbnail view:**

  By default, the `UI/Layout/CenterPane/Search/View/` `DockedListViewConfig` element points to the `UI/Layout/CenterPane/` `Search/View/ListViewConfig` element to get only the first listed field and display its name in docked List view. The field is defined in the first `<field>` property, as follows:

```
<field>
    <fieldname>fieldname</fieldname>
    <displayname>DisplayName</displayname>
```

  To display any other field name in the docked List view, specify that name in your custom `DockedListViewConfig.jsp` element. The same logic applies to displaying a field name in docked Thumbnail view (except that your configuration elements are named `ThumbnailViewConfig` and `DockedThumbnailViewConfig`).

71.3 Customizing Undocked Views

You customize the undocked List and Thumbnail views by configuring your own identically named elements and placing them under `CustomElements`.

Customizing the undocked List and Thumbnail views involves overriding the system-defined elements shown in Table 71-1.

Topics:

- Basic Steps for Customizing Undocked Views

- Setting the Default Undocked View to List or Thumbnail

- Customizing the Undocked List View

- Customizing the Undocked Thumbnail View

71.3.1 Basic Steps for Customizing Undocked Views

To customize an undocked view, take any combination of the following steps:

- Set the default undocked view to be List or Thumbnail for all asset types or your choice of asset types. To set the view(s), override the element `UI/Layout/CenterPane/Search/SearchResultsConfig`, as shown in Setting the Default Undocked View to List or Thumbnail.

- Configure the undocked List or Thumbnails views or both. Specify the number of columns to be displayed in the view(s), configure column names and column widths, specify the sort order of returned items, and more.

  - To configure the List view, override the element `UI/Layout/CenterPane/Search/View/ListViewConfig`, described in Customizing the Undocked List View.

  - To configure the Thumbnail view, override the element `UI/Layout/CenterPane/Search/View/ThumbnailViewConfig`, described in Customizing the Undocked Thumbnail View.

- Configure additional features, such as sort menus for the views. In this step, configure JSP elements that are specific to the features of the view (such as a sort menu), rather than the view itself. See Customizing Sort Menus and Tooltips.

71.3.2 Setting the Default Undocked View to List or Thumbnail

When setting the default search view (List or Thumbnail), set it globally for all asset types. You also can specify a default search view for selected asset types of your choice.

- To set the default search view(s), override the element `UI/Layout/CenterPane/Search/SearchResultsConfig` by creating your own `SearchResultsConfig.jsp` under `CustomElements` and customizing its properties.

  The `UI/Layout/CenterPane/Search/SearchResultsConfig` element is shown next. Then the table describes the properties.

  Element UI/Layout/CenterPane/Search/SearchResultsConfig

```
<searchconfig>
<listview>UI/Layout/CenterPane/Search/View/ListView</listview>
    <thumbnailview>UI/Layout/CenterPane/Search/View/ThumbnailView</thumbnailview>
    <defaultview>listview</defaultview>
    <assettypeviews>
        <assettype id="Page" name="Page">listview</assettype>
          …
          …
          …
    </assettypeviews>
</searchconfig>
```

Table 71-5   Properties in `UI/Layout/CenterPane/Search/SearchResultsConfig`

Property	Description	Value
`<listview>`	Path to the ListView controller element.	`UI/Layout/CenterPane/Search/View/ListView` Do not change the value of this property.
`<thumbnailview>`	Path to the ThumbnailView controller element.	`UI/Layout/CenterPane/Search/View/ThumbnailView` Do not change the value of this property.
`<defaultview>`	Specifies whether List or Thumbnail is the default view. The default view is the view that opens the first time search is run. If the user switches the view, search remembers the user's choice until browser cookies are cleared.	`listview | thumbnailview` The value of this property is case-sensitive.
`<assettypeviews>`	Used to selectively configure a default view for one or more asset types.	N/A
`<assettype id= name= >`	Used to specify the asset type and its default view (which remains until the user either switches to a different view or clears browser cookies). You can specify as many asset types as necessary (one per `<assettype>`).	`<assettype id="unique_identifier" name="AssetTypeName"> listview | thumbnailview </assettype>`

71.3.3 Customizing the Undocked List View

When customizing the List view, you can set the type of content to be returned and its presentation.

- To customize the undocked List view, override the `UI/Layout/CenterPane/Search/View/ListViewConfig` element by creating your own `ListViewConfig.jsp` under `CustomElements` and customizing its properties.

  The `UI/Layout/CenterPane/Search/View/ListViewConfig` element is shown next. Then the table describes the properties.

  Element UI/Layout/CenterPane/Search/View/ListViewConfig

```
<listviewconfig>
    <numberofitems>1000</numberofitems>
    <numberofitemsperpage>100</numberofitemsperpage>
    <defaultsortfield>  </defaultsortfield>
    <defaultsortorder>  </defaultsortorder>
    <fields>
      <field id="name">
        <fieldname>name</fieldname>
        <displayname>Name</displayname>
        <width>350px</width>
```

```
                              <formatter>fw.ui.GridFormatter.nameFormatter</formatter>
                              <displayintooltip>true</displayintooltip>
                  </field>
                  <field id="updateDate">
                              <fieldname>updateddate</fieldname>
                              <displayname>Modified</displayname>
                              <!-- <dateformat>MM/dd/yyyy hh:mm a z </dateformat> -->
                              <javadateformat>SHORT</javadateformat>
                              <width>auto</width>
                              <formatter></formatter>
                              <displayintooltip>true</displayintooltip>
                   </field>
                      …
                      …
                      …
                   </fields>
          </listviewconfig>
```

Table 71-6   Properties in `UI/Layout/CenterPane/Search/View/ListViewConfig`

Property	Description	Value
`<numberofitems>`	Maximum number of items returned by search.	Integer greater than 0. If `-1` is entered for instance, then all results matching the search criteria are returned.
`<numberofitemsperpage>`	Number of rows per page needed in the search results.	`100` is the default.
`<defaultsortfield>`	Default field that search should sort when fetching search results.	The default is empty. Therefore, search results are displayed by relevance. Configure this element if any other field should be set as the default for sorting.
`<defaultsortorder>`	Sort order used by search.	`ascending | descending` Required when `<defaultsortfield>` is specified.
`<fields>`	Columns that are shown in List view. These columns are shown in the same order as listed under `<fields>`. If you are creating an asset type-specific configuration and you want to display asset type-specific attributes in the search results, you must enable the asset type index and attribute search. See Configuring Attributes for Asset Type Index and Adding Asset Types to the Search Index in *Oracle Fusion Middleware Administering Oracle WebCenter Sites*. If you skip this procedure, search uses the global index.	N/A
`<field id= >`	Defines a column to be shown in List view.	`<field id="unique_identifier">`

Table 71-6 (Cont.) Properties in `UI/Layout/CenterPane/Search/View/ListViewConfig`

Property	Description	Value
`<fieldname>`	Asset's field name to render in the column.	This name must match the column name in the Lucene index. If `locale` is added as the field name, then it is displayed only if the site dimension is enabled.
`<displayname>`	Display name shown in the column header.	Alphanumeric string
`<width>`	Width of the column in pixels.	Width in units of `px` (for example, `350px`). Oracle recommends setting the width to `auto` for the last field.
`<formatter>`	Dojo formatter function to display column values in your preferred format.	The formatter must be made available in a Dojo module. See the `modules` property in `UI/Config/GlobalHtml`.
`<displayintooltip>`	Indicates whether the associated field must be listed in the tooltip for docked List view. The element `UI/Layout/CenterPane/ Search/View/ SearchToolTipHtml` renders tooltips and uses the value of this property to determine whether to list the associated field name in the tooltip (the field value also is listed). Tooltips can be customized only for docked views. See Customizing Tooltips for Search Results.	`true` | `false`
`<dateformat>`	Applies to date fields only. This is an option to specify a custom date format if the date needs to be displayed in a format other than `javadateformat`.	A valid date format string. If `<dateformat>` is used, it takes precedence over `<javadateformat>`.
`<javadateformat>`	Applies to date fields only.	Valid values are `SHORT`, `MEDIUM`, `LONG`, and `FULL`. If `<javadateformat>` is omitted or left blank, the system uses `SHORT` by default. If `<dateformat>` is used, it takes precedence over `<javadateformat>`.

71.3.4 Customizing the Undocked Thumbnail View

When customizing the Thumbnail view, you can set the type of content to be returned and its presentation.

To customize the undocked Thumbnail view, override the element `UI/Layout/ CenterPane/Search/View/ThumbnailViewConfig` by creating your own

ThumbnailViewConfig.jsp under CustomElements and customizing its properties.

The UI/Layout/CenterPane/Search/View/ThumbnailViewConfig element is shown next. Then the table describes the properties.

> **Note:**
>
> Pay particular attention to the following properties: `<formatter>` and `<assettypes>`. While the element UI/Layout/CenterPane/Search/ View/ThumbnailViewConfig is mostly the same as UI/Layout/ CenterPane/Search/View/ListViewConfig, the `<formatter>` property is defined differently. Also, the `<assettypes>` property is exclusive to ThumbnailViewConfig, where it is used to render thumbnails.
>
> The `<assettypes>` property is described in detail in More About the `<assettypes>` Section in the ThumbnailViewConfig Element, where its usage is illustrated with examples. One of the examples shows how to supplement video assets with a custom element that shows a video player.

Element UI/Layout/CenterPane/Search/View/ThumbnailViewConfig

```
<thumbnailviewconfig>
    <numberofitems>1000</numberofitems>
    <defaultsortfield></defaultsortfield>
    <defaultsortorder></defaultsortorder>
    <numberofitemsperpage>12</numberofitemsperpage>
    <formatter>fw.ui.GridFormatter.thumbnailFormatter</formatter>
    <fields>
      <field id="name">
        <fieldname>name</fieldname>
        <displayname>Name</displayname>
        <displayintooltip>true</displayintooltip>
      </field>
      <field id="updateDate">
        <fieldname>updateddate</fieldname>
        <displayname>Modified</displayname>
        <!-- <dateformat>MM/dd/yyyy hh:mm a z </dateformat> -->
        <javadateformat>SHORT</javadateformat>
        <displayintooltip>true</displayintooltip>
      </field>
       …
        …
        …
    </fields>
  <assettypes>
     <assettype id="unique_identifier">
        <type>AVIImage</type>
        <subtype>Image</subtype>
        <element>UI/Layout/CenterPane/Search/View/ImageThumbnail</element>
        <attribute>imageFile</attribute>
      </assettype>
          …
          …
          …
    </assettypes>
</thumbnailviewconfig>
```

Table 71-7   Properties in UI/Layout/CenterPane/Search/View/ThumbnailViewConfig

Property	Description	Value
`<numberofitems>`	Maximum number of items to be returned by search.	Integer greater than 0. If -1 is entered for instance, then all results matching the search criteria are returned.
`<numberofitemsperpage>`	Number of rows per page needed in the search results.	100 is the default value.
`<formatter>`	Dojo formatter function to display values in your preferred format.	The formatter must be made available in a Dojo module. See the `modules` property in `UI/Config/GlobalHtml`.
`<defaultsortfield>`	Default sort field that search should sort when fetching search results.	The default is empty. Therefore, search results are displayed by relevance. Configure this element if any other field should be set as a default for sorting.
`<defaultsortorder>`	Sort order used by search.	`ascending	descending` This is required when `<defaultsortfield>` is specified.
`<fields>`	Fields that are shown below the thumbnails in Thumbnail view. These fields are shown in the same order as listed under `<fields>`. If you are creating an asset type-specific configuration and you want to display asset type-specific attributes in the search results, you must enable the asset type index and attribute search. See the following topics in *Oracle Fusion Middleware Administering Oracle WebCenter Sites*: • Adding Asset Types to the Search Index • Configuring Attributes for Asset Type Index If you skip this procedure, search uses the global index.	N/A
`<field id=>`	Describes a field under the thumbnail.	`<field id="unique_identifier">`

Table 71-7   (Cont.) Properties in `UI/Layout/CenterPane/Search/View/ThumbnailViewConfig`

Property	Description	Value
`<fieldname>`	Asset's field name to render below the thumbnail.	This name must match the column name in the Lucene index. If `locale` is added as the field name, then it is displayed only if the site dimension is enabled.
`<displayname>`	Display name to render below the thumbnail.	Alphanumeric string
`<dateformat>`	Applies to date fields only. This is an option to specify a custom date format if the date needs to be displayed in a format other than `javadateformat`.	A valid date format string. If `<dateformat>` is used, it takes precedence over `<javadateformat>`.
`<javadateformat>`	Applies to date fields only.	Valid values are `SHORT`, `MEDIUM`, `LONG`, and `FULL`. If `<javadateformat>` is omitted or left blank, the system uses `SHORT` by default. If `<dateformat>` is used, it takes precedence over `<javadateformat>`.
`<displayintooltip>`	Indicates whether the associated field must be listed in the tooltip for docked Thumbnail view. The element `UI/Layout/CenterPane/Search/View/SearchToolTipHtml` renders tooltips. It uses the value of the `<displayintooltip>` property to determine whether to list the associated field in the tooltip (the field value also is listed). Tooltips can be customized only for docked views. For instructions, see Customizing Tooltips for Search Results.	`true` | `false`
`<assettypes>`	This section specifies the asset types for which special thumbnails are shown. Each asset type must have an attribute whose content is rendered as a thumbnail. To learn when this section must be customized, see More About the `<assettypes>` Section in the ThumbnailViewConfig Element.	N/A

Table 71-7 (Cont.) Properties in `UI/Layout/CenterPane/Search/View/ThumbnailViewConfig`

Property	Description	Value
`<assettype id= >`	Describes the asset type for which a special thumbnail is shown.	`<assettype id="unique_identifier">`
`<type>`	Name of the asset type for which a thumbnail is rendered.	See Re-using the System-Defined Image Thumbnail Element and Using a Custom Thumbnail-Rendering Element.
`<subtype>`	Subtype of the asset type.	See Re-using the System-Defined Image Thumbnail Element and Using a Custom Thumbnail-Rendering Element.
`<element>`	Path to the controller element that renders the content specified in `<attribute>` as a thumbnail.	See Re-using the System-Defined Image Thumbnail Element and Using a Custom Thumbnail-Rendering Element. If you do not specify an element, the system-defined element `UI/Layout/ CenterPane/Search/View/ GlobalThumbnail` is used to render static icons, stored in the `images/ search` directory. See Use of Static Icons.
`<attribute>`	Attribute whose content is shown as a thumbnail.	See Re-using the System-Defined Image Thumbnail Element and Using a Custom Thumbnail-Rendering Element.

71.3.4.1 More About the `<assettypes>` Section in the ThumbnailViewConfig Element

Table 71-7 contains the `<assettypes>` section, which may have to be configured, depending on which features you choose to customize. Various `<assettypes>` configuration scenarios are discussed below in the context of the most commonly performed customizations.

This section includes the following topics:

- Use of Static Icons
- Re-using the System-Defined Image Thumbnail Element
- Using a Custom Thumbnail-Rendering Element

71.3.4.1.1 Use of Static Icons

To use your own static thumbnails (stored in the file system), it is not required to customize the `<assettypes>` section of the `UI/Layout/CenterPane/Search/ View/ThumbnailViewConfig` element, if you observe the following conventions:

- The name of the thumbnail icon should not contain spaces (they are replaced with underscores). The name must be in one of the following formats, depending on the size of the thumbnail:

  - `<assettypename>.png` or `<assettypename>-<subtype>.png` (small thumbnail, 96x96, docked view)

  - `<assettypename>_large.png` or `<assettypename>-<subtype>_large.png` (large thumbnail, 170x170, undocked view)

- The storage location of the icon is the `/images/search` directory of the file system.

If the above conventions are followed, the icon are automatically rendered as a thumbnail by the `UI/Layout/CenterPane/Search/View/ThumbnailViewConfig` element, which is coded to look for icons in the `/images/search` directory. Naming the icon after the asset type and subtype automatically associates the icon with assets of that type and subtype.

71.3.4.1.2 Re-using the System-Defined Image Thumbnail Element

Customizing the `<assettypes>` section of the `ThumbnailViewConfig.jsp` element is a requirement to dynamically render *custom images* as thumbnails by re-using the system-defined element `ImageThumbnailHtml.jsp`. This element processes images that are associated with image attributes belonging to specific asset types or subtypes or both.

To re-use the system-defined `ImageThumbnailHtml.jsp`, in your custom `ThumbnailViewConfig.jsp`, do the following:

1. Specify the asset types that require a custom image thumbnail. **Each asset type must have an image attribute.**

```
<assettypes>
    <assettype>
        <type>Name_of_AssetType_containing_the_image_attribute</type>
        <subtype>Name_of_subtype_containing_the_image_attribute</subtype>
        <element>UI/Layout/CenterPane/Search/View/ImageThumbnail</element>
        <attribute>Name_of_imageAttribute_containing_the_image</attribute>
    </assettype>
     …
     …
    </assettypes>
```

2. For `<element>`, specify the path to the system-defined element `ImageThumbnailHtml.jsp`, exactly as shown in the sample code above.

71.3.4.1.3 Using a Custom Thumbnail-Rendering Element

Customizing the `<assettypes>` section of the `ThumbnailViewConfig.jsp` is a requirement if you plan to use a custom element that dynamically renders the content of an asset type's (or subtype's) `blob` attribute as a thumbnail.

In the example below, you create elements that work together to render video thumbnails. The figure below shows a sample video Thumbnail view, which you can reproduce by following the steps in this topic.

Figure 71-3   Sample Video Thumbnail View
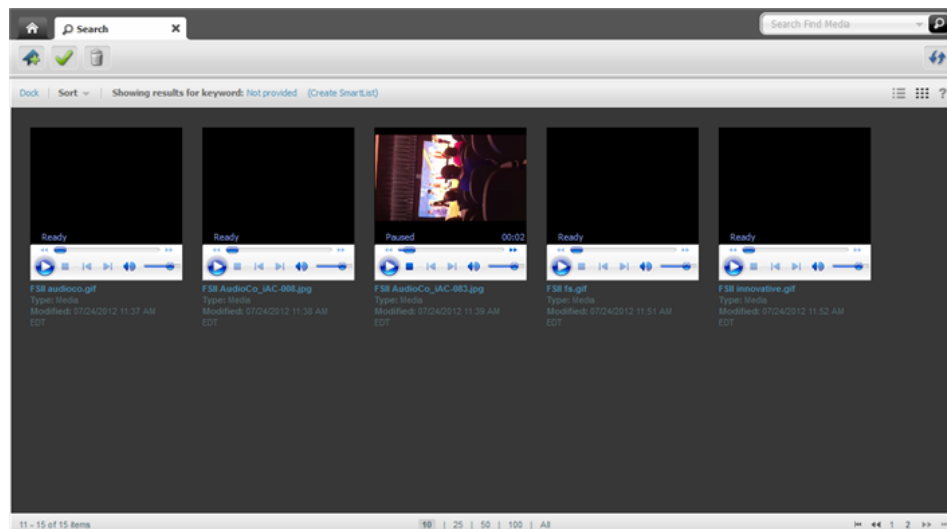
The steps below provide guidelines for (1) creating elements that work together to dynamically render video thumbnails, and (2) customizing the `<assettypes>` section of the `ThumbnailViewConfig` element.

> **Note:**
>
> To make this sample work, ensure that you have assets with a `blob` attribute and video files for that `blob` attribute are uploaded to your site's directory.
>
> To play the video file the browser plugin should be available.

To create elements that render video thumbnails:

1. Write a video thumbnail `Action` element that uses the AssetAPI and gets the URL of the `blob` using `BlobUtil` for the video attribute specified in the element. (The element can be named as you want, but it must end in `Action`. The element should be stored in a directory under `CustomElements`.)

   A sample element named `VideoThumbnailAction.jsp` is available in the zip file containing this guide.

2. Write a video thumbnail `Html` element, which takes the URL built in the previous step and renders the video and other asset details below the thumbnail. (The element can be named as you want, but it must end in `Html`. The element should be stored in a directory under `CustomElements`.) This `Html` element calls the `Action` element.

   A sample element named `VideoThumbnailHtml.jsp` is available in the zip file containing this guide.

3. To use the video thumbnail `Html` element, configure the `<assettype>` property in your custom `ThumbnailViewConfig.jsp` element as shown below:

```
<assettype>
    <type>Name_of_AssetType_containing_blob_attribute</type>
    <subtype>Name_of_asset_subtype</subtype>
    <element>CustomElements/path_to_your_element/Element</element>
```

```
        <attribute>Name_of_attribute_containing_video</attribute>
</assettype>
```

A sample element named `ThumbnailViewConfig.jsp` is available in the zip file containing this guide.

71.4 About Customizing Docked Views

Methods for customizing docked views are similar to those for undocked views. The main differences are outlined in Customization Processes.

71.5 Customizing Sort Menus and Tooltips

There are some features that you can customize for undocked views, some for docked views, and some for both.

Features discussed in this section can be customized for undocked views, docked views, or both, as shown in this table.

Table 71-8   Customizing Other Features for Search Views

Customization Option	Undocked List	Undocked Thumbnail	Docked List View	Docked Thumbnail	See …
Sort Menus	No	Yes	Yes	Yes	Customizing Sort Menus
Tooltips for Search Results	No	No	Yes	Yes	Customizing Tooltips for Search Results
Context Menus	Yes	Yes	Yes	Yes	Customizing Context Menus

Topics:

- Customizing Sort Menus
- Customizing Tooltips for Search Results
- Customizing Context Menus

71.5.1 Customizing Sort Menus

Sort menus can be customized only for the views listed in Table 71-8. You can specify which sort fields to display in a sort menu. You also can specify sort order for each field.

- To customize a Sort menu, override the element `UI/Layout/CenterPane/Search/View/SearchTopBarConfig` by creating your own `SearchTopBarConfig.jsp` under `CustomElements` and customizing its properties.

  The `UI/Layout/CenterPane/Search/View/SearchTopBarConfig` element is shown next. The table that follows describes the properties.

  Element UI/Layout/CenterPane/Search/View/SearchTopBarConfig

```
<sortconfig>>
     <sortfields>
```

```
                            <sortfield id="unique_identifier">
                                <fieldname>name</fieldname>
                                <displayname>Name(A-Z)</displayname>
                                <sortorder>ascending</sortorder>
                            </sortfield>
                            <sortfield id="unique_identifier">
                                <fieldname>name</fieldname>
                                <displayname>Name(Z-A)</displayname>
                                <sortorder>descending</sortorder>
                            </sortfield>
                            <sortfield id="unique_identifier">
                                <fieldname>AssetType_Description</fieldname>
                                <displayname>Asset Type</displayname>
                                <sortorder>ascending</sortorder>
                            </sortfield>
                                …
                                …
                        </sortfields>
                </sortconfig>
```

Table 71-9   Properties in `UI/Layout/CenterPane/Search/View/SearchTopBarConfig`

Property	Description	Value
`<sortfield id= >`	Describes the search index field by which to sort search results.	`id=unique_identifier`
`<fieldname>`	Name of the search index field. The same field can be repeated multiple times to provide multiple sort orders.	For example, `name` in the code above.
`<displayname>`	Display name of the user-readable field.	For example, `Name` in the code above.
`<sortorder>`	Sort order.	`ascending` | `descending`

71.5.2 Customizing Tooltips for Search Results

Tooltips can be customized only for docked views. Docked views are displayed in a limited space and therefore provide a limited amount of information about the assets that are returned as search results. Tooltips are a way of displaying more information about the returned assets. For example, you can customize tooltips to display field names and values in addition to those displayed in docked mode, as shown in the figure below. You can also customize tooltips to display custom messages, and you can modify the appearance of tooltips.

Figure 71-4   Tooltip in Undocked List View
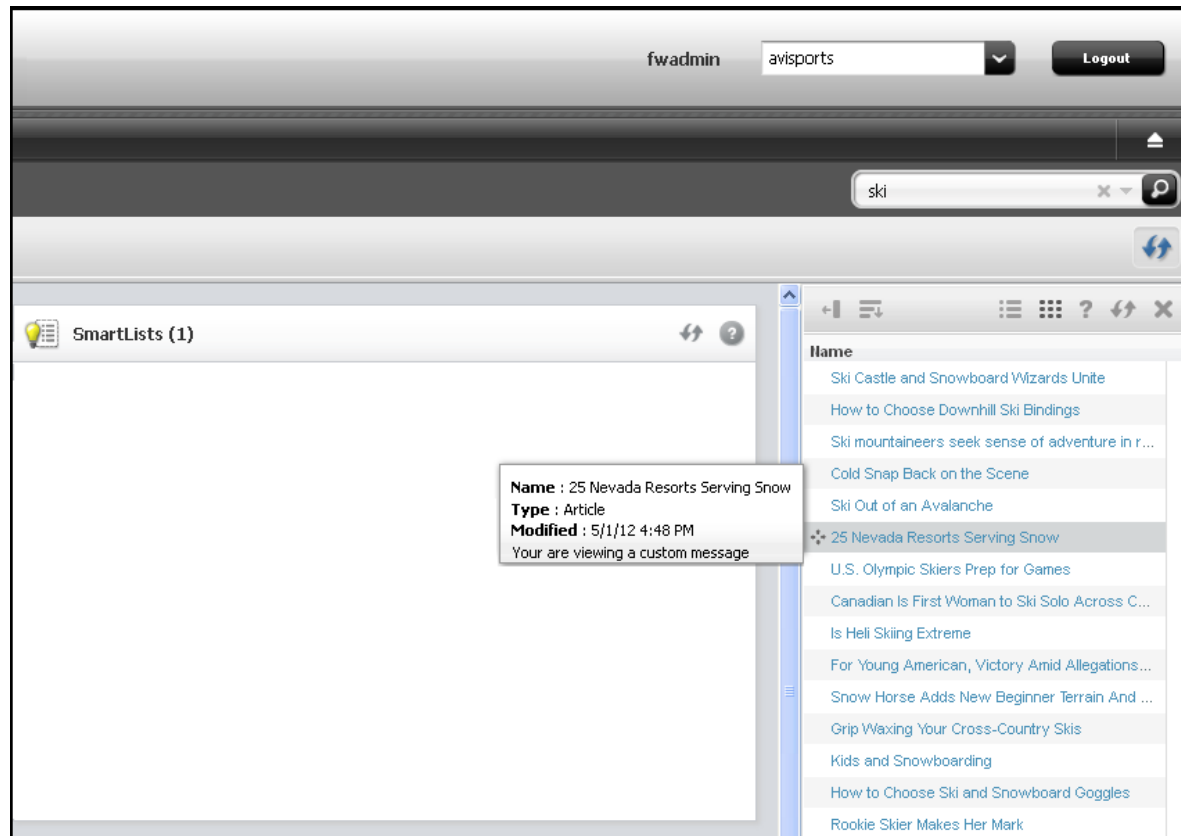
The default tooltip for docked search results is rendered by the element `UI/Layout/CenterPane/Search/View/SearchToolTipHtml`. This element renders the tooltip as a box. Within the box, it renders the name of each field in the `<fields>` section of `UI/Layout/CenterPane/Search/View/ListViewConfig` (or `UI/Layout/CenterPane/Search/View/ThumbnailViewConfig`), but only if the field's `<displayintooltip>` property is set to `true`. For example, the `Name`, `Type`, and `Modified` fields in the `ListViewConfig.jsp` below are displayed as part of the tooltip in the preceding figure, given that `<displayintooltip>` is set to `true`:

```
<fields>
        <field>
            <fieldname>name</fieldname>
            <displayname>Name</displayname>
            <width>350px</width>
            <formatter>fw.ui.GridFormatter.nameFormatter</formatter>
            <displayintooltip>true</displayintooltip>
        </field>
        <field>
            <fieldname>type</fieldname>
                <displayname>Type</displayname>
                <width>auto</width>
                <formatter></formatter>
                <displayintooltip>true</displayintooltip>
        </field>
        <field>
                <fieldname>updateddate</fieldname>
                <displayname>Modified</displayname>
                <javadateformat>SHORT</javadateformat>
                <width>auto</width>
```

```
<formatter></formatter>
<displayintooltip>true</displayintooltip>
</field>
```

> **Note:**
>
> The `UI/Layout/CenterPane/Search/View/SearchToolTipHtml`
> element also renders field values. However, customized messages and
> changes to tooltip appearance must be coded in the custom
> `SearchToolTipHtml.jsp` element.

To create a tooltip or add fields to the tooltip:

1. To create a tooltip, override the element `UI/Layout/CenterPane/Search/View/SearchToolTipHtml` by creating your own `SearchToolTipHtml.jsp` under `CustomElements`.

2. To add fields to the tooltip, add the fields to your custom `ListViewConfig.jsp` or `ThumbnailViewConfig.jsp` and set each field's `<displayintooltip>` property to `true`.

3. To display a custom message in the tooltip (custom or system-defined) or to change the appearance of the tooltip, code a custom `SearchToolTipHtml.jsp` element. For example:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"
%><%@ taglib prefix="ics" uri="futuretense_cs/ics.tld"
%><cs:ftcs>
<style>
.customSearchTooltip {
        font-weight: bold;
        color: #333;
        font-style: italic;
}
</style>

<div class='customSearchTooltip'>
        You are Viewing a Custom Tooltip
</div>
</cs:ftcs>
```

71.5.3 Customizing Context Menus

Context menus can be customized for all search views.

To customize a context menu, override the element `UI/Config/GlobalHtml` by creating your own `MyConfig.jsp` with the following code:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld" %>
<cs:ftcs>
   webcenter.sites['${param.namespace}'] = function (config) {
     config.contextMenus = {
       "default":["bookmark"],
       "asset":["edit","preview", "share", "bookmark", "tagasset"],
       "asset/Page":["edit", "preview", "delete", "bookmark"],
       "proxy":["preview", "bookmark", "tagasset"],
       ;
```

```
      }
</cs:ftcs>
```

72

Customizing Global Properties, Toolbar, and Menu Bar in the Contributor Interface

You work with the global configuration element (`UI/Config/GlobalHtml`) to customize the global features (such as global properties, toolbar, and menu bar) of the Contributor interface.

Topics:

- Customizing Global Configuration Properties

- Customizing the Toolbar

- Customizing the Menu Bar

- Customizing Context Menus

72.1 Customizing Global Configuration Properties

Global configuration properties are used to set display conditions for the Contributor interface across all content management sites.

This section includes the following topics:

- About the Configuration Properties

- Default Configuration Properties That Can Be Modified

- Adding Custom Configuration Properties

72.1.1 About the Configuration Properties

The client-side framework retrieves its main configuration settings from the server-side controller element `UI/Config/GlobalHtml`. This presentation element serves JavaScript code, which is executed by the client-side application at startup. The JavaScript code defines a JavaScript function, whose name is given as a request parameter by the client-side application:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"%>
<cs:ftcs>
webcenter.sites['${param.namespace}'] = function (config) {
    config.maxTabCount = 50;
    config.defaultView =  …;
     … merge
}
</cs:ftcs>
```

The `config` object is then manipulated as needed in the function body, by setting the properties expected by the client-side application.

In addition, as explained below, the client-side application is capable of retrieving additional configuration properties from the server-side, which allows to merge settings from multiple sources, without having to duplicate the global properties in multiple locations.

72.1.2 Default Configuration Properties That Can Be Modified

The following describes the system-defined configuration properties and indicates which properties can be modified.

Table 72-1 Configuration Properties in `UI/Config/GlobalHtml`

Property Name	Description	Values and Examples
maxTabCount	Maximum number of tabs that can remain open simultaneously. A tab is the tab of an open asset.	Any integer greater than 0. For example: `config.maxTabCount = 30;`
enableContextMenu	Indicates whether the default browser context (right-click) menu should be enabled when users work in web mode.	true | false For example: `config.enableContextMenu = true;`
enableWebMode	Indicates whether web mode should be enabled. When this property is set to `false`, users can work only with assets in form mode and use the preview functionality. By default, this property takes the value of the `xcelerate.enableinsite` property, found in `wcs_properties.json`.	true | false For example: `config.enableWebMode = true;`
enableDatePreview	Indicates whether date-based preview should be enabled. By default, this property takes the value of the `cs.sitepreview` property, found in `wcs_properties.json`.	true | false For example: `config.enableDatePreview = false;`
enablePreview	Indicates whether preview is allowed. By default, this property takes the value of the Preview method attribute in the Edit Site form (accessible from the Admin interface: Select **Admin** tab, expand **Sites**, double-click *SampleSite*, and select **Edit**).	true | false For example: `config.enablePreview = true;`

Table 72-1   (Cont.) Configuration Properties in `UI/Config/GlobalHtml`

Property Name	Description	Values and Examples

`defaultView`	Defines the preferred view for working with assets (that is, whether assets are viewed, by default, in form mode or web mode).
	
	Note: An asset is opened in web mode only if the asset is associated with a default template.
		The expected value is one of the following:
		
		 • `"default": "form"` | `"web"`
		
		 • `"assetType" :"form"` | `"web"`
		
		 • `"assetType/subtype":` `"form"` | `"web"`
		
		 where `assetType` is a valid asset type name, and `subtype` is a valid subtype or definition name. For example:
		
		 `config.defaultView = {`
		 `"default": "form",`
		 `"AVIArticle": "web",`
		 `"Page/AVISection": "web"`
		 `}`

`toolbars`	Defines the list of available toolbar actions for each type of view.	See Customizing the Toolbar.
`toolbarButtons`	Used to define the behavior of specific toolbar buttons.	See Customizing the Toolbar with Custom Actions.
`menubar`	Defines the list of available actions in the menu bar.	See Customizing the Menu Bar.
`documents`	Registers available implementations of documents.	Do not modify the value of this property. The only supported value is `asset`.
`views`	Registers view implementations.	Do not modify the value of this property.
`controllers`	Registers controller implementations and the set of actions supported by each controller.	Do not modify the value of this property.
`roles`	Contains the list of roles for the currently logged in user.	Do not modify the value of this property.
`supportedTypes`	Contains the list of asset types that can be edited from the Contributor interface.	Do not modify the value of this property.
`searchableTypes`	Contains the list of asset types that can be searched from the Contributor interface.	Do not modify the value of this property.
`token`	Used for security when uploading binary file.	Do not modify the value of this property.
`sessionid`	Used for security when uploading binary file.	Do not modify the value of this property.

Table 72-1   (Cont.) Configuration Properties in `UI/Config/GlobalHtml`

Property Name	Description	Values and Examples
contextMenus	Defines the list of available actions in the context menu.	See Customizing Context Menus.

72.1.3 Adding Custom Configuration Properties

In addition to retrieving the global properties, stored in `UI/Config/GlobalHtml`, the Contributor application attempts to retrieve additional settings in `UI/Config/SiteConfig` and any element present in `UI/Config`. Depending on the requirement, this lets you set global properties, or site-specific properties, without having to replicate all the properties defined in `UI/Config/GlobalHtml`, but only the properties that actually change.

This section includes the following topics:

- Adding Custom Global Properties

- Adding Site-Specific Properties

72.1.3.1 Adding Custom Global Properties

Custom global properties are meant to be shared across all sites on a given content management system. The recommended approach consists of creating a custom configuration element defined as follows:

- The presentation element name must be `UI/Config/SiteConfigHtml`.

- The element code must follow the pattern shown in About the Configuration Properties.

For example, you may want to:

- Override the value of `maxTabCount` for all sites.

- Override the default view for Page assets.

- And, define an additional custom property called `foo`.

To do this, create an element called `CustomElements/UI/Config/SiteConfigHtml`, containing the following code:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld" %>
<cs:ftcs>
webcenter.sites['${param.namespace}'] = function (config) {
    // override existing properties
    config.maxTabCount = 60;
    config.defaultView.Page = "form";

    // add custom properties
    config.foo = "bar";

}
</cs:ftcs>
```

72.1.3.2 Adding Site-Specific Properties

In some cases, the Contributor interface must be configured differently for each content management site. It is recommended that you override the core controller element called `UI/Config/SiteConfig`.

To override the `UI/Config/SiteConfig` core element, create an element as follows:

```
CustomElements/siteName/UI/Config/SiteConfigHtml
```

where `siteName` is the name of the content management site (for instance, `avisports`).

For example, the avisports demo site enforces web mode as the default mode for assets of type Page and AVIArticle. This is done by defining the JSP element `CustomElements/avisports/UI/Config/SiteConfigHtml`, and providing the following settings:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld" %>
<%@ taglib prefix="ics" uri="futuretense_cs/ics.tld" %>
<cs:ftcs>
webcenter.sites['${param.namespace}'] = function (config) {
    // default view modes for avisports
    config.defaultView.Page = "web";
    config.defaultView.AVIArticle = "web";
}
</cs:ftcs>
```

Loading of Configuration Elements

The global configuration element is always loaded first. Additional configuration elements are loaded in alphabetic order. For instance, using the examples above, configuration properties would be loaded in the following order:

1. `UI/Config/GlobalHtml`

2. `UI/Config/SiteConfigHtml`

Property Values

The value of some properties is, in some cases, an object. That is:

```
config.someProperty = {
  foo: "bar",
  x: 123
};
```

When partially overriding this property, it is important to distinguish between the following types of code:

```
config.someProperty = {
  x: 3456
};
```

vs.

```
config.someProperty.x = 3456;
```

In the first case, the property `foo` is overridden as "undefined", whereas in the second case, the original value of `foo` is preserved.

72.2 Customizing the Toolbar

On the toolbar you can list actions for operating on assets in web mode or form mode. You can customize the toolbar further per asset type and subtype.

Topics:

- About Toolbar Customization

- Examples of Toolbar Customization

72.2.1 About Toolbar Customization

The global configuration element (`UI/Config/GlobalHtml`) describes for each type of view (such as web mode inspect, web mode edit, form mode edit, and form mode inspect), the list of actions to display in the toolbar to the user. This is done through the `toolbars` property. Its value is an object with the following syntax:

```
config.toolbars = {
    "viewAlias": [action_1, action_2,  …],
    or:
    "viewAlias": {
        "view_mode_1": [action_1, action_2,  …],
        "view_mode_2": [action_1, action_2,  …]
    }
 …
}
```

where:

`viewAlias` indicates for which type of view this toolbar must be used. The alias must match one of the view aliases defined in the `config.views` section.

`action_i` is an action name. For standard actions, such as `save` and `approve`, the action name is automatically mapped to a given icon, title, alternate text, and so on. For more information about standard actions, custom actions, or customizing the appearance of a custom button, see Examples of Toolbar Customization.

`view_mode_i` is one of the modes supported by the view (typically, `edit` or `view`).

72.2.2 Examples of Toolbar Customization

This section includes the following topics:

- Customizing the Toolbar with Standard Actions for Web Mode

- Customizing the Toolbar with Standard Actions for Asset Type and Subtype

- Customizing the Toolbar with Custom Actions

72.2.2.1 Customizing the Toolbar with Standard Actions for Web Mode

- Use the following configuration which determines the toolbar actions that are available in web mode for all asset types:

```
config.toolbars = {
    ( …)

    "web": {
      "edit": ["form-mode", "inspect", "separator", "save", "preview",
```

```
                      "approve", "delete", "separator", "changelayout",
                      "separator", "checkincheckout", "refresh"],
            "view": ["form-mode", "edit", "separator", "preview", "approve",
                      "delete", "separator", "checkincheckout", "refresh"]

          ( …)

    }
```

The above configuration defines two lists of actions (`edit` and `view`), corresponding to the asset's views: Edit and Inspect.

Note:

To find the set of standard actions, refer to the list of actions specified in the following properties under the `controllers` property:

- `fw.ui.document.AssetDocument` (all actions supported by assets)

- `fw.ui.controller.InsiteController` (all actions supported by the view controller)

72.2.2.2 Customizing the Toolbar with Standard Actions for Asset Type and Subtype

Each toolbar configuration can be customized by asset type and subtype by adding a property named:

- *viewAlias/assetType*

- *viewAlias/assetType/assetSubtype*

For example, add the **Bookmark**/**Unbookmark** buttons for page assets in web mode.

- In a custom configuration element (such as `CustomElements/avisports/UI/Config/SiteConfigHtml`), add the following property:

```
config.toolbars["web/Page/AVISection"] = {
    "edit": config.toolbars.web.edit, // reuse default for edit mode
    "view": ["form-mode", "edit", "separator", "preview", "approve", "delete",
                "bookmark", "unbookmark", "separator",
                "checkincheckout",  "refresh"]
}
```

Inspecting the Surfing Page asset now shows the toolbar.

Figure 72-1   Surfing Page Asset Toolbar
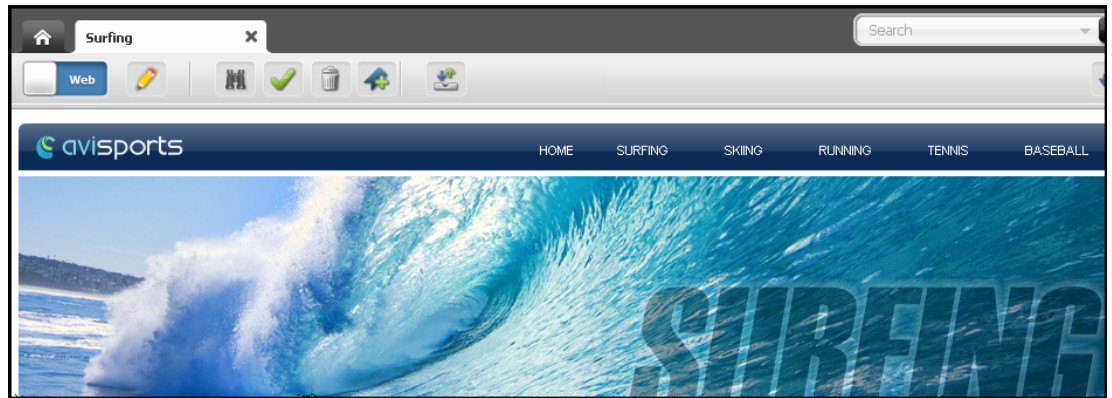
Note:

Keep in mind the following:

- We are customizing only the `view` mode. When a Page/AVISection asset is being edited in web mode, the standard toolbar is shown.

- The **Bookmark** and **Unbookmark** buttons are not shown simultaneously, because they both depend on the asset's current state (whether it is bookmarked or not).

72.2.2.3 Customizing the Toolbar with Custom Actions

- To define custom actions add new entries to the `config.toolbarButtons` property, as follows:

```
config.toolbarButtons.<customActionName> = {
  src: <path_to_icon>,
  onClick: <click_handler>
}
```

- For example, let's define the following `helloWorld` custom action:

```
config.toolbarButtons.helloWorld = {
    src: 'js/fw/images/ui/ui/toolbarButton/smartlist.png',
    onClick: function () {
        alert('Hello World!!');
    },
    buttonType: 'button'
}
```

The `helloWorld` action can now be referenced from a toolbar configuration as follows (we reuse our example from the previous section):

```
config.toolbars["web/Page/AVISection"] = {
    "view":
        ["form-mode", "edit", "separator", "preview", "approve",
         "bookmark", "unbookmark", "separator",
         "checkincheckout", "separator", "helloWorld", "refresh"],

    "edit": config.toolbars.web.edit // reuse default web mode toolbar
}
```

> **Note:**
>
> In this example, we have added a separator (a vertical line) and the custom button to the toolbar.

Figure 72-2   Separator and Toolbar Button
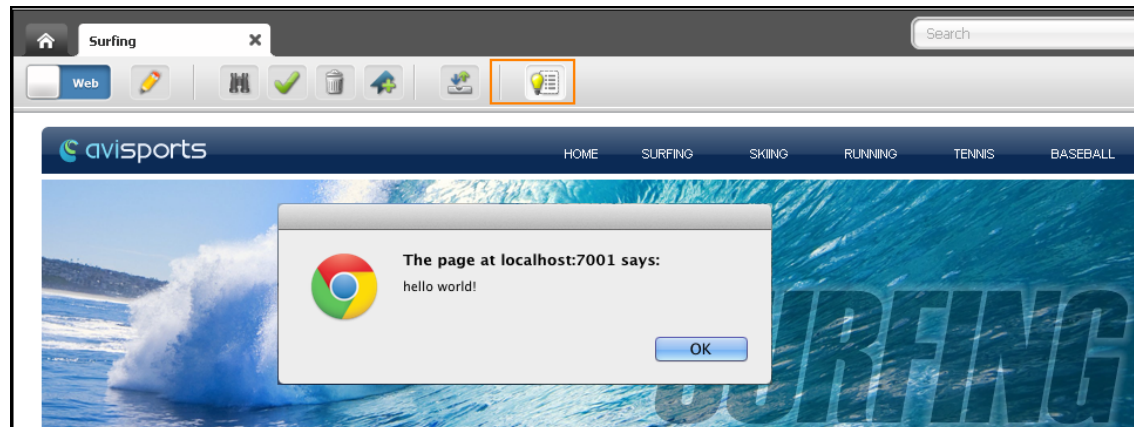
A more elaborate example would involve, for instance, creating a CSElement which outputs an asset's id and type, and then creating a toolbar option that calls the element when clicked. For example:

1. Create a CSElement (HelloAsset, in this example) which outputs assets' id and type.

2. Now, define the helloAsset custom action by adding new entries to the config.toolbarButtons property (in the SitesConfigHTML.jsp), as follows:

```
config.toolbarButtons.helloAsset = {
      src: 'js/fw/images/ui/ui/toolbarButton/smartlist.png',
      onClick: function () {
       /* current active document which holds the asset*/
        var doc = SitesApp.getActiveDocument();
        var asset = doc.get('asset');
        var id = asset.get("id");
        var type = asset.get("type");
        /* make an ajax call to call an element */
        dojo.xhrGet({
            url: "url of element to call", /*url of the element*/
            /*pass any parameters that need to be passed to the element */
            content:{"id":id,"type":type}
            /*handleAs:"json" */ /* this is needed only if the element returns
json response */
            }).then(function(response) {
                console.log("Response:", response);
                /*handle response from the element here */
                alert(response);
          },
            function(err){
                /*handle error*/
                console.log("error on ajax call");
          });
      }
};
```

3. Add the custom button (defined in step 2) to the toolbar under `config.toolbars` ={ }. For example:

```
/*add the custom button in web mode for AVIArticle*/
      config.toolbars["web/AVIArticle"] = {
            "view" : ["form-mode", "edit", "separator", "preview", "approve",
"delete",
            "bookmark", "unbookmark", "separator", "checkincheckout",
"helloAsset",
            "separator", "refresh",]
      };
```

This figure shows the custom option in the toolbar:

Figure 72-3   Custom helloAsset Toolbar Button

When the custom `helloAsset` button is clicked, the `CSElement` (created in step 1) is called and a dialog box opens providing the asset's name, id, and type:

Figure 72-4   Dialog Box Showing Asset's Name, ID, and Type
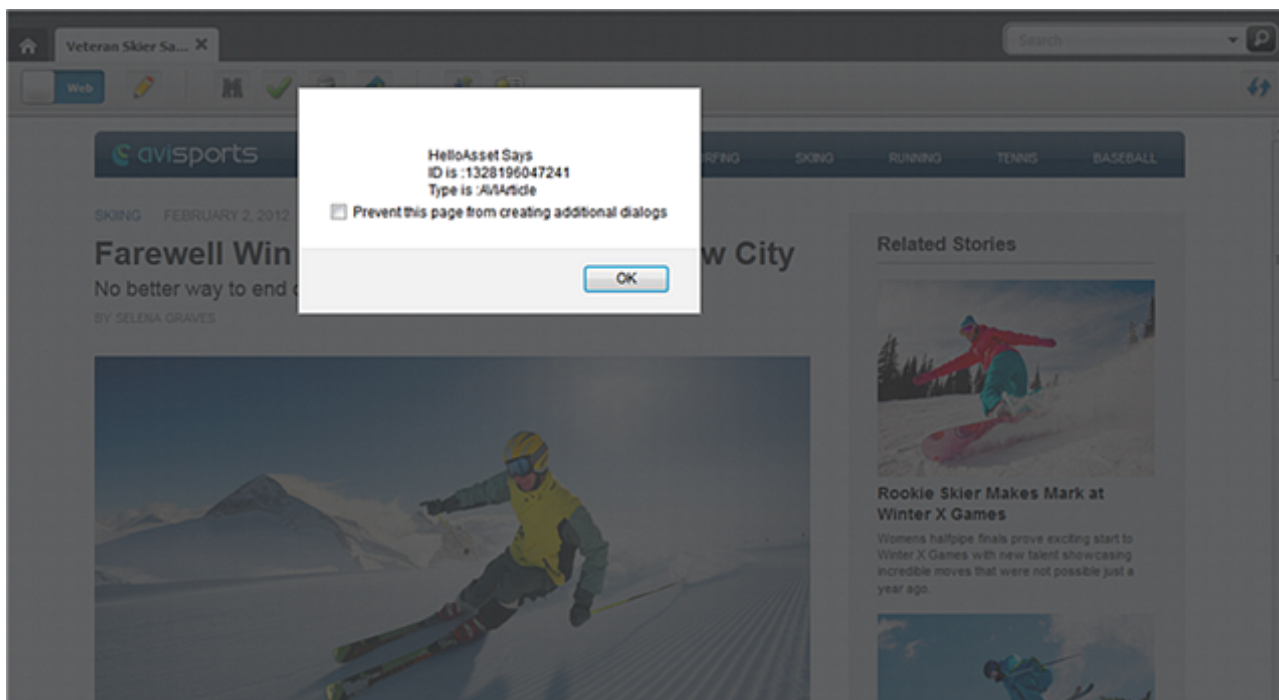
72.3 Customizing the Menu Bar

On the menu bar, you can add submenus that let contributors operate on assets of a certain type and subtype. You can make these submenus actionable items, additional menus, or menu separators.

Topics:

- About Menu Bar Customization

72.3.1 About Menu Bar Customization

The menu bar configuration is defined by the `config.menubar` property:

```
config.menubar = {
        "key_i": [
                //menu_i
                {
                        "id": "menu_id",
                        "label": "menu_label",
                        "children": [
                            //submenus
                            //- actionable menu item
                                {
                                        label: 'menu_item_label',
                                        action: 'action_name' | click_handler
                                },
                                //- deferred pop-up menu
                                {
                                        label: 'menu_item_label',
                                        deferred: 'controller_element',
                                        cache: true|false
                                },
                                //- pop-up menu
                                {
                                        label: 'menu_item_label',
                                        children: [
                                                // submenu_1
                                                {
                                                        label:
'menu_item_label',

                                                        action: 'action_name' |
click_handler
                                                }'
                                                // submenu_2
                                                {
                                                        label:
'menu_item_label',

                                                        action: 'action_name' |
click_handler
                                                },
                                                 …
                                                 …
                                                 …
                                        ]
                                },
                                //- menu item separator
                                {separator: true}

                        //additional menu_i
            …
            …
            …
            ]
                }
```

where:

key_i is one of the following:

- *default*: Defines the default menu bar.

- *assetType*: Defines the customized menu bar for all assets of type *assetType*.

- *assetType/subtype*: Defines the customized menu bar for all assets of type *assetType* and subtype *subtype*.

//menu_i starts a section that describes each top menu, where:

- *menu_id* is the identifier of the menu.

- *menu_label* is the display name of the menu.

- submenus can be any of the following:

- – An **actionable menu item** (clicking the menu item produces an action), where:

  * label specifies the display name of the menu item.

  * action can be any action supported by a controller, such as edit and inspect, or a custom click handler (see the customization example in Adding a Custom Action to the Menu Bar).

> **Note:**
>
> When a given action is not supported by the current document/view, it is disabled (greyed out) in the menu.

For example, a menu item triggering a save action is defined as follows:

```
{
    label: "Save",
    action: "save"
}
```

- A **deferred pop-up menu** (the pop-up menu is determined dynamically by running a controller element on the server-side), where:

  * label specifies the display name of the menu item.

  * deferred specifies a controller element name, such as UI/Data/ StartMenu/New.

  * cache is a Boolean value indicating whether the output of the controller element should be cached or not.

For instance, the **New** pop-up menu, which reads all available start menu items for the current site/user, is defined as follows:

```
{
    label: New,
    deferred: "UI/Data/StartMenu/New",
    cache: true
}
```

- A **pop-up menu** (the child menu items are hard wired in the configuration itself).

– A **menu item separator** (a horizontal line), which is used to group menu entries together.

72.3.2 Adding a Custom Action to the Menu Bar

In this example, we want to add the `helloWorld` custom action defined in Customizing the Toolbar with Custom Actions to the menu bar (to run the custom onClick handler). Add this action by adding a new entry to the menu bar called **Custom Menu**, with a single menu item called **Hello World**, which triggers the custom action. Our steps are the following:

1. First, reuse the default menu bar, and add to it. The simplest way to do this is to make a copy of the original array:

```
config.menubar["Page/AVISection"] = config.menubar["default"].slice(0);
```

2. Add the menu, as follows:

```
config.menubar["Page/AVISection"].push(
    "id": "myCustomMenu",
    "label": "Custom Menu",
    "children": [
        // Children go here
    ]
);
```

3. Define the child menu items:

```
config.menubar["Page/AVISection"].push({
    "id": "myCustomMenu",
    "label": "Custom Menu",
    "children": [{
      "label": "Hello World",
      "action": function () {
        alert("Hello from the top menubar!");
      }
    }]
});
```

The **Custom Menu** can now be seen whenever an **AVISection** Page section is viewed.

Figure 72-5   Custom Menu
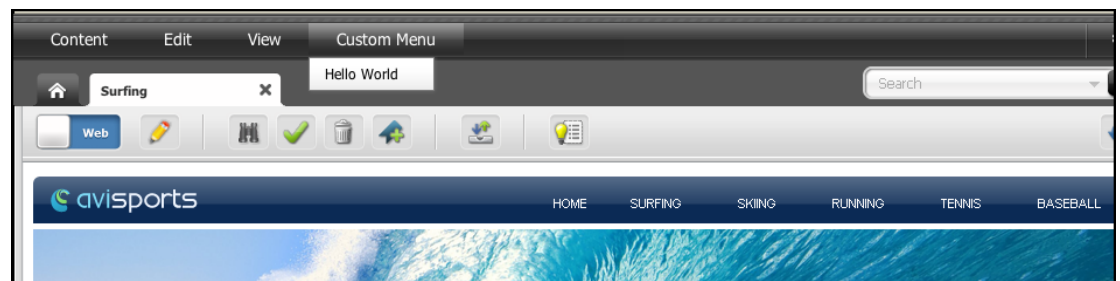
Selecting the **Hello World** menu item should run the custom onClick handler.

Figure 72-6   Custom onClick Handler
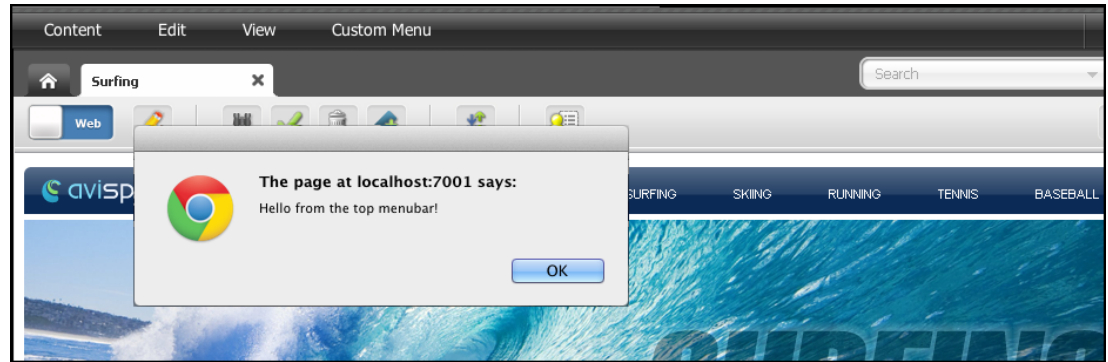
4. To run the exact same code, whether clicked from the menu bar or toolbar, write the following:

```
// define the helloWorld code once
config.myActions = {
    hello: function (args) {
      var doc = SitesApp.getActiveDocument(),
        asset = doc.get('asset'),
        view = SitesApp.getActiveView();

      view.info('Hello World!! The asset is a ' + asset.type +     ' with id:
          + asset.id);
    }
};

// attach it to the helloWorld button
config.toolbarButtons['helloworld'] = {
    src: 'js/fw/images/ui/ui/toolbarButton/smartlist.png',
    onClick: config.myActions.hello
};

config.toolbars["web/Page/AVISection"] = {
    "edit": config.toolbars.web.edit, // reuse default for edit mode
    "view": [  "form-mode", "edit", "separator", "preview", "approve",
    "delete", "bookmark", "unbookmark", "separator",
    "checkincheckout","separator","helloworld", "refresh"]
}

// attach it to the menubar, under "Custom Menu">"Hello World"
config.menubar['Page/AVISection'] = config.menubar['default'].slice(0);

config.menubar["Page/AVISection"].push({
    "id": "myCustomMenu",
    "label": "Custom Menu",
    "children": [{
        "label": "Hello World",
        "action": config.myActions.hello
    }]
});
```

72.4 Customizing Context Menus

On a right-click, a context menu shows options from which contributors choose what they need to work with the objects they right clicked. You can customize the specific items available in a context menu and the order in which the objects are displayed.

To define context menus use the `config.contextMenus` property:

```
config.contextMenus = {
        "default": ["action_1", "action_2", … "action_n"]
    "asset": ["action_1", "action_2", … "action_n"],
    "asset/assetType": ["action_1", "action_2", … "action_n"],
}
```

where the values are defined as:

- `default`: Defines the default list of context menus. Each menu item displayed in the list is an `action`.

- `asset`: Defines the customized list of context menus for all assets. Each menu item displayed in the list is an `action`.

- `asset/assetType`: Defines the customized list of context menu items for the assets of type `assetType`. Each menu item displayed in the list is an `action`.

An example context menu configuration for the asset type Page:

```
"asset/Page":["edit", "copy", "preview", "delete", "bookmark", "tagasset"]
```

73

Customizing Asset Forms for the Contributor Interface

When you customize asset forms in the Oracle WebCenter Sites: Contributor interface, you can also modify the asset form headers and also build single-valued and multi-valued attribute editors for some supported data types.

Topics:

- About Asset Forms Customization
- Modifying the Header of Asset Forms
- Building an Attribute Editor

73.1 About Asset Forms Customization

You can modify an asset form's header and customize or build an attribute editor.

See Customizing Attribute Editors and Building an Attribute Editor.

> **Note:**
>
> Unlike other components of the Contributor interface, asset forms are not in the Contributor framework. Therefore, requests for asset forms are not processed by the UI Controller.

73.2 Modifying the Header of Asset Forms

You can modify the header of an asset form by creating a custom assettype-specific element and including additional stylesheets or JavaScript code.

To modify the header of an asset form, create a custom assettype-specific element in the `OpenMarket/Xcelerate/AssetType/<AssetTypeName>/` directory.

You can include additional stylesheets or JavaScript code instead of modifying the body of the HTML pages. The name of the element must be `Header`.

73.3 Building an Attribute Editor

You can create a custom attribute editor for the data types supported in WebCenter Sites. You can also customize the look and feel of some existing ready-to-use attribute editors.

See Customizing Attribute Editors.

This topic describes how to build a custom attribute editor that supports a single value of data type `text`, `string`, `integer`, or `money`. This section also provides pointers

and sample code for implementing a multi-valued attribute editor for the same data types.

> **Note:**
>
> To create a custom attribute editor for the `blob` or `asset` data type, base your implementation on the `UPLOADER` attribute editor for the `blob` type and the `PICKASSET` attribute editor for the `asset` data type.

Topics:

- Creating a Dojo Widget and its Template

- Defining the Attribute Editor as a Presentation Object

- Creating the Attribute Editor Element

- Creating the Attribute Editor

- Implementing a Multi-Valued Attribute Editor

73.3.1 Creating a Dojo Widget and its Template

This section describes how to create a Dojo widget to handle a single value of data type `text`, `string`, `integer`, or `money`.

This section includes the following topics:

- Create a Template for the Dojo Widget

- Creating a Dojo Widget

73.3.1.1 Create a Template for the Dojo Widget

To create an HTML template for the Dojo widget:

1. In your WebCenter Sites installation directory, navigate to the `<context_root>/js/` directory.

2. Create a new directory structure under the `js` directory as follows: `extensions/dijit/templates`.

3. In the `<context_root>/js/extensions/dijit/templates` directory, create an HTML template file and give it a meaningful name; for example: `MyWidget.html`.

4. In the HTML template file, define the look and feel of the new Dojo widget. The content of this HTML template would look similar to the following:

```
<div>
   <div>
      <input type="text" dojoAttachPoint='inputNode'
       name='${name}'size='60' class='valueInputNode'>
      </input>
   </div>
</div>
```

If you use this code for the template, then the input node takes the input from the end user and the value of the input node is maintained in the Dojo widget which you create in Creating a Dojo Widget.

5. Save your template file.

73.3.1.2 Creating a Dojo Widget

To create a Dojo widget:

1. Navigate to the `js/extensions/dijit` directory of the WebCenter Sites installation.

2. Create a `dojo` widget, for example, `MyWidget.js` (see the example below) by implementing the following mandatory functions:

- `_setValueAttr`: This `setter` method sets the value of the attribute.

- `_getValueAttr`: This `getter` method gets the attribute value.

- `isValid`: This method runs validations to see if the given value is valid or not.

- `focus`: This sets the focus on the attribute editor.

- `onChange`: This method is called whenever the user updates the value of the attribute.

- `onBlur`: This method updates the widget when the attribute value is entered by the user. An update is triggered when the user selects another field.

```
dojo.provide('extensions.dijit.MyWidget');
dojo.require('dijit._Widget');
dojo.require('dijit._Templated');
dojo.declare('extensions.dijit.MyWidget', [dijit._Widget, dijit._Templated], {
  //string.
  //The value of the attribute.
  value: '',
  //int
  //The Attribute editor's MAXALLOWEDCHARS should be assigned to this variable.
  //maxAllowedLength: 15,
  //string
  //  This variable is required only for single valued instance.
  //  The server should recieve information from input element with this name.
  name: '',
  //HTMLElement
  //  This stores the cached template of the widget's representation.
  templateString: dojo.cache('extensions.dijit', 'templates/MyWidget.html'),
  //string
  //  This class will be applied to the top div of widget.
  //  It will help in managing css well.
  baseClass: 'MyWidget',
  postCreate: function() {
      var self = this;
      // Do not allow typing characters more than allowed length.
        dojo.connect(this.inputNode, 'onkeypress', function(e) {
        if (this.value.length >= self.maxAllowedLength && e.keyCode !=
            dojo.keys.BACKSPACE)
        e.preventDefault();
      });
},
// Start -  Mandatory functions
_setValueAttr: function(value) {
```

```
      //  summary:
      //     Set the value to 'value' attribute and input node
      if (value === undefined || !this._isValid(value)) return;
      this.value = value;
      this._setInputNode(value);
},
_getValueAttr: function() {
   // summary:
   //Get the latest value and return it.
   return this.value;
},
_isValid: function(newVal) {
   //summary:
   //Verify if the given value is as per the expectation or not.
   if (newVal.length > this.maxAllowedLength) {
   return false;
}
   return true;
},
   focus: function() {
   //summary:
   //Set the focus to the representation node, that is, input node here.
   if (typeof this.inputNode.focus === 'function')
   this.inputNode.focus();
},
onBlur: function() {
   //summary:
   //Custom selected browser event when the value should be updated
   //Any activity which leads to value change should update
   //the widget value as well.
   this.updateValue();
},
_onChange: function(newValue) {
   //summary:
   //Internal onChange method
   this.onChange(newValue);
},
onChange: function(newValue) {
   //summary:
   //A public hook for onChange.
},
   // End -  Mandatory functions
   // Extra functions used in Mandatory functions
_setInputNode: function(value) {
  //summary:
  //Sets the value to input node.
  this.inputNode.value = value;
},
updateValue: function() {
  //summary:
  //Validate the newly entered value and if it is successful
  //then update widget's value.
  var newVal = this.inputNode.value;
  if (!this._isValid(newVal)) return;
  if (this.value != newVal)
  this._onChange(newVal);
  this.set('value', newVal);
}
});
```

> **Note:**
>
> For information about creating Dojo widgets, see:
>
> http://dojotoolkit.org/

3. Create the `js/extensions/themes/`directory, create a CSS (for example, `MyWidget.css`) for this widget. Use the following code in the CSS file, or write your own code:

```
.fw .MyWidget .valueInputNode {
color: blue;
}
```

4. In the `js/extensions/themes/`directory, create the `UI.css` with an import statement for the Dojo widget's CSS. For example, `@import url("MyWidget.css");`

5. Save your work.

73.3.2 Defining the Attribute Editor as a Presentation Object

This section describes how to define input tags (presentation objects) for flex attributes. It also describes how to assign arguments that the input tags can pass from the attribute editor to the display elements.

To define the attribute editor:

1. In your WebCenter Sites installation, navigate to the `<OracleHome>\wcsites \webcentersites\sites-home\sites.war\WEB-INF\sites \presentationobject.dtd` file, or to the `<OracleHome>\wcsites \webcentersites\sites-home\sites-samples.war\WEB-INF\sites \presentationobject.dtd` file (whichever path applies to your installation).

2. In the `presentationobject.dtd` file, do the following;

   - Add a new tag (presentation object) to the list in the `<!ELEMENT PRESENTATIONOBJECT ...>` statement. In this example, the new tag is named `MYATTREDITOR`.

     In the following line, `MYATTREDITOR` is the custom attribute editor whose name matches the name of the element you create in Creating the Attribute Editor Element. All other tags are ready-to-use attribute editors.

     ```
     <!ELEMENT PRESENTATIONOBJECT (TEXTFIELD | TEXTAREA | PULLDOWN | RADIOBUTTONS
     | CHECKBOXES | PICKFROMTREE | EWEBEDITPRO | REMEMBER |
     PICKASSET | FIELDCOPIER | DATEPICKER | IMAGEPICKER | REALOBJECT |
     CKEDITOR | DATEPICKER | IMAGEPICKER | REALOBJECT | CKEDITOR | FCKEDITOR |
     UPLOAD | MAGEEDITOR | RENDERFLASH | PICKORDERASSET | TYPEAHEAD | UPLOADER |
     MYATTREDITOR)>
     ```

   - Add an `<!ELEMENT ... >` section that defines the new tag (presentation object) and the arguments it takes. This new tag includes elements that supply the logic behind the format and behavior of the attribute when it is displayed on a form. Ensure that `MAXALLOWEDCHARS` is marked as a required attribute.

     ```
     <!ELEMENT MYATTREDITOR ANY>
     <!ATTLIST MYATTREDITOR MAXALLOWEDCHARS CDATA #REQUIRED>
     <!ATTLIST MYATTREDITOR MAXVALUES CDATA #IMPLIED>
     ```

- Save and close the `presentationobject.dtd` file.

73.3.3 Creating the Attribute Editor Element

This section describes how to create an element that shows an "edit" view of an attribute (single-valued) when it displays in a New or Edit form. This element must be located in the `OpenMarket/Gator/AttributeTypes` directory in the `ElementCatalog` table. The element name must exactly match the name of the tag you defined in Defining the Attribute Editor as a Presentation Object, so that it can be invoked by the tag (in this example, `MYATTREDITOR`).

1. Navigate to the `OpenMarket/Gator/AttributeTypes` directory in your ElementCatalog.

2. Create an attribute element for your new editor (in this example, `MYATTREDITOR.jsp`.) Ensure that the name of this element matches the tag name you defined in the `presentationobject.dtd` file.

3. To prevent the default rendering of the attribute editor, set the `doDefaultDisplay` variable to `no`.

4. To display the attribute name, call the element `OpenMarket/Gator/FlexibleAssets/Common/DisplayAttributeName`. The code is:

```
<ics:callelement
  element="OpenMarket/Gator/FlexibleAssets/Common/DisplayAttributeName"/>
```

5. To render the widget, call the element `OpenMarket/Gator/AttributeTypes/CustomTextAttributeEditor` by using the following parameters:

- `editorName`: Name of the widget created in Creating a Dojo Widget. In this example it is `extensions.dijit.MyWidget`.

- `editorParams`: This argument passes the JSON string of parameters to the widget. In this example, it passes the `maxAllowedLength` value. For example, the value can look like this: `{ maxAllowedLength: "10" }`

- `maximumValues`: Required only for a multi-valued widget. This is the maximum number of values allowed to be rendered in a multi-valued widget.

    For a single-valued widget, the complete code with the initialization parameters and formatting styles should look like the code in the following code example.

If you use the code given in the following example, then the single-valued attribute editor would look like the editor.

Figure 73-1   Single-Valued Attribute Editor
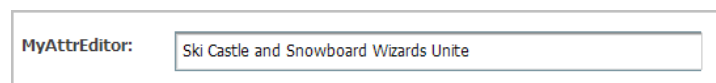
Note:

In the example below, the following core logic is implemented to render the single-valued attribute using the new attribute editor:

```
<ics:if condition='<%= "no".equals(ics.GetVar("MultiValueEntry")) %>'>
<ics:then>
      <div dojoType='<%= ics.GetVar("editorName") %>'
      name='<%= ics.GetVar("cs_SingleInputName") %>'
      value='<%= attributeValue %>'
      >
      </div>
</ics:then>
```

The name that is coded in the element must be `ics.GetVar("cs_SingleInputName")` to ensure that the input node in the Dojo template has the same name. The input node value is sent to the server for saving the attribute.

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld" %>
<%@ taglib prefix="ics" uri="futuretense_cs/ics.tld" %>
<%@ taglib prefix="satellite" uri="futuretense_cs/satellite.tld" %>
<%//
// OpenMarket/Gator/AttributeTypes/MYATTREDITOR
//
// INPUT
//
// OUTPUT
//%>
<%@ page import="COM.FutureTense.Interfaces.FTValList" %>
<%@ page import="COM.FutureTense.Interfaces.ICS" %>
<%@ page import="COM.FutureTense.Interfaces.IList" %>
<%@ page import="COM.FutureTense.Interfaces.Utilities" %>
<%@ page import="COM.FutureTense.Util.ftErrors" %>
<%@ page import="COM.FutureTense.Util.ftMessage"%>
<cs:ftcs>
<ics:setvar name="doDefaultDisplay" value="no" />
<script>
      dojo.require('extensions.dijit.MyWidget');
</script>
<link href="<%=ics.GetVar("cs_imagedir")%>/../js/extensions/themes/MyWidget.css"
      rel="stylesheet" type="text/css"/>
<%
FTValList args = new FTValList();
args.setValString("NAME", ics.GetVar("PresInst"));
args.setValString("ATTRIBUTE", "MAXALLOWEDCHARS");
args.setValString("VARNAME", "MAXALLOWEDCHARS");
ics.runTag("presentation.getprimaryattributevalue", args);
args.setValString("NAME", ics.GetVar("PresInst"));
args.setValString("ATTRIBUTE", "MAXVALUES");
args.setValString("VARNAME", "MAXVALUES");
ics.runTag("presentation.getprimaryattributevalue", args);
String maximumValues = ics.GetVar("MAXVALUES");
maximumValues = null == maximumValues ? "-1" : maximumValues;
String editorParams = "{ maxAllowedLength: "
      + ics.GetVar("MAXALLOWEDCHARS") + " }";
%>
<tr>
<ics:callelement
```

```
     element="OpenMarket/Gator/FlexibleAssets/Common/DisplayAttributeName"/>
       <td></td>
       <td>
       <ics:callelement
         element="OpenMarket/Gator/AttributeTypes/CustomTextAttributeEditor">
           <ics:argument name="editorName" value="extensions.dijit.MyWidget" />
           <ics:argument name="editorParams" value='<%= editorParams %>' />
           <ics:argument name="maximumValues" value="<%= maximumValues %>" />
</ics:callelement>
       </td>
  </tr>
</cs:ftcs>
```

73.3.4 Creating the Attribute Editor

This topic describes how to create an attribute editor asset to make it available to content contributors on their content management sites. This asset supports the input types you defined in Creating the Attribute Editor Element, for example, check boxes, radio options, and drop-down lists. The developer selects this editor when creating or modifying the attribute.

1. Open the Admin interface of your site.

2. On the **New** page, under the **Name** column, click **New Attribute Editor**.

3. For **Name**, enter a meaningful name for your editor. For example, `MyAttrEditor`.

4. For **XML**, enter the XML code for your attribute editor. Ensure that the name of the attribute editor in this code is exactly the same as the element name. For example:

```
<?XML VERSION="1.0"?>
<!DOCTYPE PRESENTATIONOBJECT >
<PRESENTATIONOBJECT NAME="MYATTREDITOR">
<MYATTREDITOR MAXALLOWEDCHARS="10"> </MYATTREDITOR> </PRESENTATIONOBJECT>
```

5. For **Attribute Type**, accept the appropriate value(s).

6. Click the **Save** icon.

The attribute editor similar to the editor in the following figure is created.

Figure 73-2   Sample Attribute Editor for a Site


```
Attribute Editor: MyAttrEditor

       *Name:     MyAttrEditor

  Description:

      Status:     Created

          ID:     1345007628877

        Site:     FirstSite II

         XML:     <?XML VERSION="1.0"?> <!DOCTYPE PRESENTATIONOBJECT >
                  <PRESENTATIONOBJECT NAME="MYATTREDITOR"> <MYATTREDITOR
                  MAXALLOWEDCHARS="10"> </MYATTREDITOR> </PRESENTATIONOBJECT>

  Value Type:     ANY

     Created:     Tuesday, August 14, 2012 1:29:50 PM IST by fwadmin

    Modified:     Tuesday, August 14, 2012 1:29:50 PM IST by fwadmin
```

73.3.5 Implementing a Multi-Valued Attribute Editor

In Creating the Attribute Editor Element, the second example shows the implementation for a single-valued attribute editor.

- To implement a multi-valued attribute editor for `text`, `integer`, `string`, or `money` data types, write a code similar to the code in the following example:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld" %>
<%@ taglib prefix="ics" uri="futuretense_cs/ics.tld" %>
<%@ taglib prefix="satellite" uri="futuretense_cs/satellite.tld" %>
<%//
// OpenMarket/Gator/AttributeTypes/CustomTextAttributeEditor
//
// INPUT
//
// OUTPUT
//%>
<%@ page import="COM.FutureTense.Interfaces.FTValList" %>
<%@ page import="COM.FutureTense.Interfaces.ICS" %>
<%@ page import="COM.FutureTense.Interfaces.IList" %>
<%@ page import="COM.FutureTense.Interfaces.Utilities" %>
<%@ page import="COM.FutureTense.Util.ftErrors" %>
<%@ page import="COM.FutureTense.Util.ftMessage"%>
<cs:ftcs>
<%
IList attributeValueList = ics.GetList("AttrValueList", false);
boolean hasValues = null != attributeValueList && attributeValueList.hasData();
String attributeValue = hasValues ? attributeValueList.getValue("value") : "";
%>
<ics:if condition='<%= "no".equals(ics.GetVar("MultiValueEntry")) %>'>
<ics:then>
        <div dojoType='<%= ics.GetVar("editorName") %>'
                name='<%= ics.GetVar("cs_SingleInputName") %>'
```

```
              value='<%= attributeValue %>'
      >
    </div>
  </ics:then>
  <ics:else>
  <ics:callelement
    element="OpenMarket/Gator/AttributeTypes/RenderMultiValuedTextEditor">
      <ics:argument name="editorName"
        value='<%= ics.GetVar("editorName") %>' />
      <ics:argument name="editorParams"
        value='<%= ics.GetVar("editorParams") %>' />
      <ics:argument name="multiple"
        value="true" />
      <ics:argument name="maximumValues"
        value='<%= ics.GetVar("maximumValues") %>' />
  </ics:callelement>
  </ics:else>
  </ics:if>
  </cs:ftcs>
```

If the code in the example above is used, then the multi-valued attribute editor looks similar to the editor in this figure.

Figure 73-3   Multi-Valued Attribute Editor
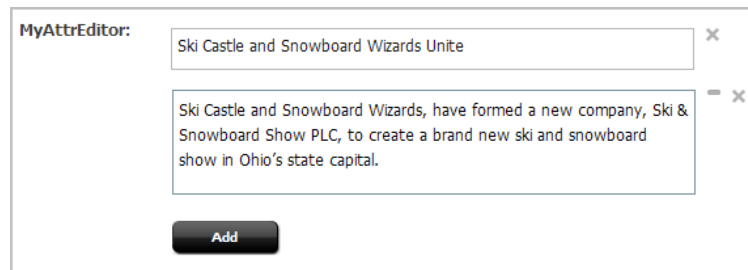
Note the following points about the code example:

– You can instantiate a multi-valued widget, which uses a single-valued widget to render multi-valued representations.

– The `MultiValueEntry` variable with the `no` value indicates that the attribute editor renders a single value. Changing the variable value to `yes` enables the attribute editor to render multiple values.

– You can implement a multi-valued widget that accepts values in the JSON object or in any other format.

– For a multi-valued attribute editor, the `RenderMultiValuedTextEditor` element creates hidden input nodes required for `Save` logic. The value of each node is sent to the server.

– The multi-valued widget is rendered by calling the `OpenMarket/Gator/ AttributeTypes/RenderMultiValuedTextEditor` element using the following code in the previous example:

```
<ics:else>
  <ics:callelement
    element="OpenMarket/Gator/AttributeTypes/RenderMultiValuedTextEditor">
      <ics:argument name="editorName"
       value='<%= ics.GetVar("editorName") %>' />
      <ics:argument name="editorParams"
```

```
                value='<%= ics.GetVar("editorParams") %>' />
            <ics:argument name="multiple"
             value="true" />
            <ics:argument name="maximumValues"
              value='<%= ics.GetVar("maximumValues") %>' />
        </ics:callelement>
    </ics:else>
    </ics:if>
```

• To create a custom attribute editor for the `blob` or `asset` data type, base your implementation on the `UPLOADER` attribute editor for the `blob` type and the `PICKASSET` attribute editor for the `asset` data type.

74

Customizing Workflow

A WebCenter Sites workflow process is the series of states an asset moves through on its way to publication. The asset moves from one state to the next by taking a workflow step. You must create the workflow step condition elements which specify the conditions that an asset must meet to move on to the next state, and the workflow action elements which perform various actions as the asset moves from one state to the next.

Topics:

- Workflow Step Conditions
- Workflow Actions

74.1 Workflow Step Conditions

A workflow process is composed of one or more workflow states. Workflow steps move the asset from one workflow state to the next. You can associate each workflow step that the asset takes with a timed action, such as sending an email to a user when an asset is assigned to them, or a workflow step condition, which prevents an asset from moving on to the next step if certain conditions are not fulfilled. Sometimes there are conditions under which the asset should not move on to the next workflow state. You need to create the element that defines the condition or conditions that prevent the asset from moving on to the next state.

This element receives the following data when it is called:

- An `IWorkflowable` object called `Object`, which represents the asset whose state is being changed.
- An `IWorkflowStep` object called `Step`, which represents the current workflow step.
- The `StepUser` variable, which contains the ID of the user attempting the step.
- Variables specified as name-value pairs when a `StepCondition` is defined in the WebCenter Sites user interface. See Setting Up the Actions and Conditions in *Oracle Fusion Middleware Administering Oracle WebCenter Sites*.

The workflow step condition element should check for a condition and return a Boolean value. If the value is false, then the step does not proceed.

The following example comes from a sample workflow step condition element:

```
<?xml version="1.0" ?>

<!DOCTYPE FTCS SYSTEM "futuretense_cs.dtd">

<FTCS Version="1.1">
```

```
<!-- OpenMarket/Xcelerate/Actions/Workflow/StepConditions/ExampleStepCondition

-

- INPUT

-

- OUTPUT

-

-->

<csvar NAME="This step condition will check if step can be taken"/><br/>
```

The first line in the following example sets an empty `ReturnVal` variable:

```
<setvar NAME="ReturnVal" VALUE="Variables.empty"/>

<!--change the value of ReturnVal to a non-empty string later on, if you want to
stop the step --> <!-- most of the stuff below are debugging statements and also
show you some items available to you to set up a condition for stopping the step-->
```

The second line in the following example uses the `WORKFLOWABLEASSET.GETDISPLAYABLENAME` tag to get the name of the asset that is in workflow:

```
<!-- get asset -->

<WORKFLOWABLEOBJECT.GETDISPLAYABLENAME OBJECT="Object"
VARNAME="assetdisplayablename"/>

Object:<csvar NAME="Variables.assetdisplayablename"/><br/>
```

Line 2 in the following example creates a variable called `StepUser` which contains the ID of the user attempting to take the step. Line 3 uses the `USERMANAGER.GETUSER` tag to load the user's ID into the `StepUser` variable. Line 4 uses the `CCUSER.GETNAME` tag to retrieve a human-readable user name, and line 5 uses the `csvar` tag to display that user name:

```
<!-- get userid -->

Userid: <csvar NAME="Variables.StepUser"/><br/>

<USERMANAGER.GETUSER OBJVARNAME="myUserObj" USER="Variables.StepUser"/>

<CCUSER.GETNAME NAME="myUserObj" VARNAME="uname"/>

Username: <csvar NAME="Variables.uname"/><br/>
```

Line 2 in the following example uses the `WORKFLOWSTEP.GETID` tag to get the ID of the current workflow step. The `WORKFLOWSTEP.GETNAME` tag, used in line 4, loads the step with the specified name:

```
<!-- getstep -->

<WORKFLOWSTEP.GETID NAME="Step" VARNAME="sid"/>

Stepid: <csvar NAME="Variables.sid "/>
```

```
<WORKFLOWSTEP.GETNAME NAME="Step" VARNAME="sname"/>

Stepname: <csvar NAME="Variables.sname"/><br/><br/>
```

The following example defines the conditions that stop the change of step from taking place. The `forcestop` and `notalloweduser` variables that the conditionals check were set as arguments when the sample step condition was defined in the WebCenter Sites interface. In a real step condition, you would test for the condition of your choice here. For example, seeing whether an article asset has an associated image.

```
<!-- This is the actual condition to stop the step. The following is just an
example. -->

<if COND="Variables.forcestop=true">

<then>

<setvar NAME="ReturnVal" VALUE="You can not take this step because forcestop=true"/>

</then>

<else>

<if COND="Variables.uname=Variables.notalloweduser">

<then>

<setvar NAME="ReturnVal" VALUE="You are not allowed to take this step"/>

</then>

</if>

</else>

</if>

</FTCS>
```

74.2 Workflow Actions

As an asset moves through workflow, it can trigger a *workflow action*. A workflow action can do anything from send an email to alert a user that he has a new asset to evaluate to breaking a deadlock after a specified period of time has elapsed.

There are five types of workflow actions:

- Step actions, which are executed as part of a transition between workflow states.

- Timed actions, which are triggered by deadlines when the asset is in a given state, thus associating the asset with a specific assignment.

- Deadlock actions, which are executed when an asset needs a unanimous vote to move to the next state, but the voters differ on which step the asset should take. The deadlock action is run whenever users choose different steps for the asset to move to.

- Group deadlock actions, which are executed when the assets in a workflow group need a unanimous vote to move to the next state, but the voters choose different steps, creating a deadlock.

- Delegation actions, which are executed when an asset is delegated. The delegated asset remains in its current workflow state, but is assigned to a new user.

Your workflow administrator must first define workflow actions using the WebCenter Sites user interface. Then you must create the elements that accomplish these workflow actions. WebCenter Sites provides several sample workflow action definitions for you to look at. See Setting Up the Actions and Conditions in *Oracle Fusion Middleware Administering Oracle WebCenter Sites*.

The following topics describe sample workflow action elements:

- Step Action Elements

- Timed Action Elements

- Deadlock Action Elements

- Group Deadlock Action Elements

- Delegation Action Elements

74.2.1 Step Action Elements

A Step Action element receives the following data when it is called:

- A `WorkflowEngine` object called `WorkflowEngine`.

- An `ObjectTotal` variable, which represents the total number of assets whose state is being changed.

- An `IWorkflowable` object called `Objectnnn`, which represents the assets whose state is being changed. nnn is a number between 0 and ObjectTotal -1.

- An `IWorkflowStep` object called `Step`, which represents the workflow step being considered.

- A `StepTargetUser` variable, which is a comma-separated list of the step's target users.

- A `StepUser` variable, which contains the ID of the user attempting the step.

- A `Group` variable, which contains the ID of the workflow group to which the assets belong (if you are using workflow groups).

- Any variables that your workflow administrator has created in the definition for this Step Action.

In the following example, the Step Action element approves assets for publish; most other Step Action elements send an email to the assignees.

```
<?xml version="1.0" ?>

<!DOCTYPE FTCS SYSTEM "futuretense_cs.dtd">

<FTCS Version="1.1">

<!-- OpenMarket/Xcelerate/Actions/Workflow/StepActions/ApproveForPublish

-

- INPUT
```

```
- Variables.ObjectTotal - number of loaded workflowasset objects

- Object[n] - loaded workflowasset objects, where n = 0 - Variables.ObjectTotal

-targets - one or more comma separated names of PubTargets for which to approve the
asset

-

- OUTPUT

-

-->

<!-- This is an action element called by step actions ApproveForPublish-->

This step action element will approve an asset for publish.<br/>
```

Line 2 in the following example uses the SETCOUNTER tag to create a counter which keeps track of the number of assets to approve. Lines 3 through 9 use the LOOP tag to loop through the assets and retrieve the asset types and IDs.

```
<!-- get the id and assettype of the asset(s) to approve -->

<SETCOUNTER NAME="count" VALUE="0"/>

<LOOP COUNT="Variables.ObjectTotal">

<WORKFLOWASSET.GETASSETTYPE OBJECT="ObjectCounters.count" VARNAME="assettype"/>

<WORKFLOWASSET.GETASSETID OBJECT="ObjectCounters.count" VARNAME="assetid"/>

<SETVAR NAME="idCounters.count" VALUE="Variables.assetid"/>

<SETVAR NAME="typeCounters.count" VALUE="Variables.assettype"/>

<INCCOUNTER NAME="count" VALUE="1"/>

</LOOP>
```

In the following example, the STRINGLIST tag is used to create a comma-separated list of publish target names. The PUBTARGET.LOAD and PUBTARGET.GET tags are used to load information about the publish targets from the PubTarget table. This information and information about the assets to be approved are passed to the ApprovePost element for further processing.

```
<!-- approve for each destination -->

<STRINGLIST NAME="publishTargets" STR="Variables.targets" DELIM=","/>

<if COND="IsList.publishTargets=true">

<then>

<LOOP LIST="publishTargets">

<PUBTARGET.LOAD NAME="pubtgt" FIELD="name" VALUE="publishTargets.ITEM"/>

<if COND="IsError.Variables.errno=false">
```

```
<then>

Approving for publish to <CSVAR NAME="publishTargets.ITEM"/><br/>

<PUBTARGET.GET NAME="pubtgt" FIELD="id" OUTPUT="pubtgt:id"/>

<CALLELEMENT NAME="OpenMarket/Xcelerate/PrologActions/ApprovePost">

<ARGUMENT NAME="targetid" VALUE="Variables.pubtgt:id"/>

<ARGUMENT NAME="assetTotal" VALUE="Counters.count"/>

</CALLELEMENT>

</then>

<else>

Cannot approve for publish to destination: <CSVAR NAME="publishTargets.ITEM"/>,
Error: <CSVAR NAME="Variables.errno"/>

</else>

</if>

</LOOP>

</then>

<else>

Cannot approve for publish. This step action requires a targets argument with one or
more comma separated publishing destination names.

</else>

</if>

</FTCS>
```

74.2.2 Timed Action Elements

Timed Action elements receive the following data when they are called:

- A WorkflowEngine object called WorkflowEngine.

- A WorkflowAssignmentTotal variable, which contains the total number of assignments for which this action applies.

- An IWorkflowAssignment object called WorkflowAssignmentnnn, which represents assignments to apply the action to. nnn is a number greater than zero.

- An optional Group variable, which contains the ID of the workflow group to which the assets belong (if you are using workflow groups).

- Any variables that your workflow administrator has created in the definition for this Timed Action.

The following excerpt is from a Timed Action element that sends an email. The text of the subject and body of this email are set in the Workflow Email forms that you access

from the **Admin** node in the Admin interface. The body text expects the following variables:

- `Variables.assetname`, which contains the name of the current asset

- `Variables.assigner`, which is the name of the user who completed the previous state in the workflow process

- `Variables.instruction`, which is the text that the assigner puts in the `Action to Take` text box as he or she completes an assignment

In the following example, the variables in the email object, `subject` and `body`, are replaced by their values:

```
<!-- translate subject -->
<EMAIL.TRANSLATESUBJECT NAME="emailobject" PARAMS="assetname=Variables.assetname"
VARNAME="subject"/>
<!-- translate body -->
<EMAIL.TRANSLATEBODY NAME="emailobject"
PARAMS="assetname=Variables.assetname&#38;time=Variables.time" VARNAME="body"/>
<!-- send mail -->
<sendmail TO="Variables.EmailAddress" SUBJECT="Variables.subject"
BODY="Variables.body"/>
</THEN>
<ELSE>
Email address: None<br/>
</ELSE>
</IF>
<inccounter NAME="COUNT" VALUE="1"/>
</loop>
</then>
</if>
</FTCS>
```

74.2.3 Deadlock Action Elements

Deadlock Action elements receive the following data when they are called:

- A `WorkflowEngine` object.

- An `ObjectTotal` variable, which represents the total number of deadlocked assets.

- An `IWorkflowable` object called `Objectnnn`, which represents the deadlocked assets.

- An `IWorkflowStep` object called `Step`, which represents the workflow step.

- A `StepTotal` variable, which contains the number of steps chosen by individual users.

- A `StepUser` variable, which contains the ID of the user attempting the step.

- An optional `Group` variable, which contains the ID of the workflow group to which the assets belong (if you are using workflow groups).

- Any variables that your workflow administrator has created in the definition for this Deadlock Action.

The following Deadlock Action element sends an email to the users who approve the asset.

The text of the subject and body of this email are set in the Workflow Email forms in the administrative user interface. The body text expects the following variables:

```
<?xml version="1.0" ?>

<!DOCTYPE FTCS SYSTEM "futuretense_cs.dtd">

<FTCS Version="1.1">

<!-- OpenMarket/Xcelerate/Actions/Workflow/DeadlockActions/SendEmailToAssignees

-

- INPUT

-

- OUTPUT

-

-->

<!-- This is an action element called by step actions SendAssignmentEmail and
SendRejectionEmail-->

<csvar NAME="This deadlock action element will send emails"/><br/>
```

Line 2 in the following example uses the EMAILMANAGER.LOAD tag to load an email object.

```
<!-- load email object -->

<EMAILMANAGER.LOAD NAME="Variables.emailname" OBJVARNAME="emailobject"/>
```

Lines 1 through 9 in the following example create a NumOfSteps variable, which contains either the total number of assets being delegated or zero.

```
<!-- get total steps -->

<if COND="IsVariable.StepTotal=true">

<then>

<setvar NAME="NumOfSteps" VALUE="Variables.StepTotal"/>

</then>

<else>

<setvar NAME="NumOfSteps" VALUE="0"/>

</else>

</if>

<removevar NAME="Step"/>

<setvar NAME="Header" VALUE="The following users have chosen the corresponding steps
```

```
that has resulted in a deadlock. Please take appropriate actions to resolve
deadlock:"/>
```

```
<setvar NAME="Message" VALUE="Variables.empty"/>
```

The code in the following three examples loop through the list of users who have put the asset in deadlock, creating an email for each one. Line 10 in the following code uses the USERMANAGER.GETUSER tag to load the user information of the user specified in the ID. Lines 11 and 12 use CCUSER tags to get the user's name and email address.

```
<!-- For each assignment object, get asignee -->
```

```
<setcounter NAME="COUNT" VALUE="0"/>
```

```
<if COND="Variables.NumOfSteps!=0">
```

```
<then>
```

```
<loop FROM="0" COUNT="Variables.NumOfSteps">
```

```
<!-- get assigner -->
```

```
<setvar NAME="userid" VALUE="Variables.StepUserCounters.COUNT"/>
```

```
<!-- get email address --->
```

```
<USERMANAGER.GETUSER USER="Variables.userid" OBJVARNAME="userobj"/>
```

```
<CCUSER.GETNAME NAME="userobj" VARNAME="user_name"/>
```

```
<CCUSER.GETEMAIL NAME="userobj" VARNAME="EmailAddress"/>
```

Lines 1 through 6 in the following example use the WORKFLOWSTEP and WORKFLOWSTATE tags to retrieve the asset's starting and ending steps and states.

```
<WORKFLOWSTEP.GETNAME NAME="StepCounters.COUNT" VARNAME="stepname"/>
```

```
<WORKFLOWSTEP.GETSTARTSTATE NAME="StepCounters.COUNT" VARNAME="startstate"/>
```

```
<WORKFLOWSTEP.GETENDSTATE NAME="StepCounters.COUNT" VARNAME="endstate"/>
```

```
<WORKFLOWSTATE.GETSTATENAME NAME="Variables.startstate" VARNAME="startstatename"/>
```

```
<WORKFLOWSTATE.GETSTATENAME NAME="Variables.endstate" VARNAME="endstatename"/>
```

```
<setvar NAME="Message" VALUE="Variables.Message Variables.user_name:
Variables.stepname - "/>
```

```
<!--
```

```
user:<csvar NAME="Variables.user_name"/><br/>
```

```
step name:<csvar NAME="Variables.stepname"/><br/>
```

```
startstate name:<csvar NAME="Variables.startstate"/><br/>
```

```
endstate name:<csvar NAME="Variables.endstate"/><br/>
```

```
-->
```

```
<!-- get asset -->

<WORKFLOWABLEOBJECT.GETDISPLAYABLENAME NAME="Variables.ObjectCounters.COUNT"
VARNAME="assetname"/>
```

In lines 3 and 6 in the following example, the variables in the email object, `subject` and `body`, are replaced by their values.

```
<!-- translate subject -->

<SETVAR NAME="params"
VALUE="username=Variables.user_name&#38;header=Variables.Header&#38;message=Variables
.Message&#38;assetname=Variables.assetname"/>

<EMAIL.TRANSLATESUBJECT NAME="emailobject" PARAMS="Variables.params"
VARNAME="subject"/>

<!-- translate body -->

<EMAIL.TRANSLATEBODY NAME="emailobject" PARAMS="Variables.params" VARNAME="body"/>

<!-- send mail -->

<sendmail TO="Variables.EmailAddress" SUBJECT="Variables.subject"
BODY="Variables.body"/>

<inccounter NAME="COUNT" VALUE="1"/>

</loop>

</then>

</if>

email message:<csvar NAME="Variables.Header Variables.Message"/><br/>

</FTCS>
```

74.2.4 Group Deadlock Action Elements

Group Deadlock action elements receive the following data when they are called:

- A `WorkflowEngine` object called `WorkflowEngine`

- An `ObjectTotal` variable, which represents the total number of deadlocked assets

- An `IWorkflowable` object called `Object`nnn, which represents the deadlocked asset. nnn is a number greater than zero

- An `IWorkflowStep` object called `Step`, which represents the workflow step

- A `StepTotal` variable, which contains the number of steps chosen by individual users

- A `StepUser` variable, which contains the ID of the user attempting the step

- A `Group` variable, which contains the ID of the workflow group that is deadlocked

- Any variables that your workflow administrator has created in the definition for this Group Deadlock Action

The following Group Deadlock Action element sends an email to the users who approve the asset.

The text of the subject and body of this email are set in the Workflow Email forms in the administrative user interface. The body text expects the following variables:

```
<?xml version="1.0" ?>

<!DOCTYPE FTCS SYSTEM "futuretense_cs.dtd">

<FTCS Version="1.1">

<!-- OpenMarket/Xcelerate/Actions/Workflow/GroupActions/SendEmailToAssignees

-

- INPUT

-

- OUTPUT

-

-->

<!-- user code goes here -->

<csvar NAME="This group deadlock action element will send emails"/><br/>

<!-- load email object -->

<EMAILMANAGER.LOAD NAME="Variables.emailname" OBJVARNAME="emailobject"/>

<!-- get group -->

<WORKFLOWENGINE.GETGROUPID ID="Variables.Group" OBJVARNAME="grpobj"/>

<WORKFLOWGROUP.GETNAME NAME="grpobj" VARNAME="GroupName"/>

<!-- get total steps -->

<if COND="IsVariable.StepTotal=true">

<then>

<setvar NAME="NumOfSteps" VALUE="Variables.StepTotal"/>

</then>

<else>

<setvar NAME="NumOfSteps" VALUE="0"/>

</else>

</if>

<removevar NAME="Step"/>
```

```
<setvar NAME="Header" VALUE="The following users have chosen the corresponding steps
that has resulted in a deadlock for the group: Variables.GroupName. Please take
appropriate actions to resolve deadlock:"/>

<setvar NAME="Message" VALUE="Variables.empty"/>

<!-- For each assignment object, get asignee -->

<setcounter NAME="COUNT" VALUE="0"/>

<if COND="Variables.NumOfSteps!=0">

<then>

<loop FROM="0" COUNT="Variables.NumOfSteps">

<!-- get assigner -->

<setvar NAME="userid" VALUE="Variables.StepUserCounters.COUNT"/>

<!-- get email address --->

<USERMANAGER.GETUSER USER="Variables.userid" OBJVARNAME="userobj"/>

<CCUSER.GETNAME NAME="userobj" VARNAME="user_name"/>

<CCUSER.GETEMAIL NAME="userobj" VARNAME="EmailAddress"/>

<WORKFLOWSTEP.GETNAME NAME="StepCounters.COUNT" VARNAME="stepname"/>

<WORKFLOWSTEP.GETSTARTSTATE NAME="StepCounters.COUNT" VARNAME="startstate"/>

<WORKFLOWSTEP.GETENDSTATE NAME="StepCounters.COUNT" VARNAME="endstate"/>

<WORKFLOWSTATE.GETSTATENAME NAME="Variables.startstate" VARNAME="startstatename"/>

<WORKFLOWSTATE.GETSTATENAME NAME="Variables.endstate" VARNAME="endstatename"/>

<!-- get asset -->

<WORKFLOWABLEOBJECT.GETDISPLAYABLENAME NAME="Variables.ObjectCounters.COUNT"
VARNAME="assetname"/>

<!-- set message -->

<setvar NAME="Message" VALUE="Variables.Message Asset: Variables.assetname, User:
Variables.user_name, Step: Variables.stepname -- "/>

<!-- translate subject -->

<SETVAR NAME="params"
VALUE="username=Variables.user_name&#38;header=Variables.Header&#38;message=Variables
.Message&#38;assetname=Variables.assetname"/>

<EMAIL.TRANSLATESUBJECT NAME="emailobject" PARAMS="Variables.params"
VARNAME="subject"/>

<!-- translate body -->

<EMAIL.TRANSLATEBODY NAME="emailobject" PARAMS="Variables.params" VARNAME="body"/>
```

```
<!-- send mail -->

<sendmail TO="Variables.EmailAddress" SUBJECT="Variables.subject"
BODY="Variables.body"/>

<inccounter NAME="COUNT" VALUE="1"/>

</loop>

</then>

</if>

email message:<csvar NAME="Variables.Header Variables.Message"/><br/>

</FTCS>
```

74.2.5 Delegation Action Elements

Delegation action elements receive the following data when they are called:

- A `WorkflowEngine` object called `WorkflowEngine`

- A `CurrentUser` variable, which contains the ID of the user who is delegating the asset

- An optional `Group` variable, which contains the ID of the workflow group. All objects to be delegated must be in the same workflow group

- A `DelegateUser` variable, which contains the ID of the user to whom the asset was delegated

- A `DelegateComment` variable, which contains a comment addressed to the user ID contained in the `DelegateUser` variable

- An `ObjectTotal` variable, which represents the total number of assets being delegated

- An `IWorkflowable` object called `Objectnnn`, which represents the assets being delegated. nnn represents a number greater than zero

Delegation action elements should be coded like other Workflow Action elements.

75

Working with RealTime Publishing Customization Hooks

Some of things that you need to do when you customize RealTime Publishing are writing your own transporter, writing implementation details, writing helper methods, writing example transporter implementation, writing full code listing, and writing edge-case scenarios. You may also need to write information about intercepting asset publishing events on the management instance.

You can customize RealTime publishing according to your customers' business needs. See Working with RealTime Publishing Customization Hooks and Understanding Asset and Publish Events in WebCenter Sites.

A Realtime publishing environment is configured to WebCenter Sites through the advpub.xml file. The following list provides some terminologies used to describe the publishing components:

- **Resource:** It is a generic term used to indicate approved assets and non-asset data like tables.

- **Resource group:** It is a group of resources, possibly related in some way.

Topics:

- About RealTime Publishing

- Writing a Custom Transporter

75.1 About RealTime Publishing

RealTime Publishing is a pipeline consisting of several jobs. Some jobs run on the management instance, while others on the target instance. RealTime Publishing parameters are located in the `advpub.xml` file.

The following is a brief description of each job:

- **Gatherer**: Creates the list of publishable assets and decorates it with additional resources (asset types, table rows) that together make up the canonical set of data to be published. It does so by creating groups of interdependent resources while relying on the underlying Approval Grouping Strategy. Grouping strategy obtains the lists of approved resources and organizes them into groups. Following are the grouping strategies available.

    - **ApprovalAggregatingGroupingStrategy** – This is the default grouping strategy and it is recommended to keep this strategy. It creates collection of Resource groups. In a resource group, the dominant asset type is one which has most assets of the same type in that group. While creating resource groups, this strategy keeps the count of the assets of the same asset type. The one with the maximum count is the dominant asset type for that group.

Similarly, while creating resource group collection, this strategy collects the groups with the same dominant asset types together. This strategy helps to aggregate similar groups together because loading and saving assets of the same type works faster in bulk. This strategy leads to larger number of groups with relatively smaller size.

– **ApprovalAccumulatingGroupingStrategy** – This strategy simply collects the resource groups without any additional processing. This leads to smaller number of groups with relatively larger size.

- **DataSerializer and DataDeserializer**: It serializes and deserializes the data using XStream implementation.

- **Packager**: Given the resource listing assembled by Gatherer, Packager creates serialized renditions of each resource and saves it in the local `fw_PubDataStore` table.

- **Transporter**: Takes the serialized data in `fw_PubDataStore` created by Packager and copies it to the target-side `fw_PubDataStore` table. The serialized data can be transported to multiple destinations by providing a customized multitransporter as described in the Developer's guide. The property `xcelerate.concurrenttransportunpacker` in wcs_properties.json file decides whether to run the `DataTransporter` and the `DataUnpacker` simultaneously (by default). See Code for Writing RealTime Publishing Transporter.

- **Unpacker**: Takes the serialized data in the target-side `fw_PubDataStore` table and deserializes/saves it to the target database. Although the `DataTransporter` and the `DataUnpacker` run simultaneously, the `DataUnpacker` waits until the main packaging is completed and certain priority group information is received.

  The number of `DataUnpacker` threads required to run can be configured using `numParalleltasks`. The default value for number of `DataUnpacker` threads to run or `numParallelTasks` is 1 for MSSQL server and DB2 databases and the default value is 3 for Oracle database. It is recommended to not change the default value.

- **Monitor**: Communicates and keeps track of all the messages it receives from all the participants in the publish session. These messages are stored to `PubMessage` and `PubProgress` table. `PubSessionMonitor` is a component of the asynchronous messaging system.

  Polling frequency or `pollFreqMillis` is measured in milliseconds and the default value is 5000.

  Time in milliseconds or `timeoutMillis` is the number of seconds that `PubSessionMonitor` should wait for a message before presuming that the participant has crashed or hung. Default value is 100000 (100 seconds) milliseconds.

- **CacheUpdater**: Given the list of assets that were successfully saved by Unpacker, CacheUpdater flushes and optionally regenerates relevant parts of the page caches.

  Regeneration of specified pages can be done in multiple threads based on the value of `numThreadsPerServer`. The default value of `numThreadsPerServer` is 3. The component `regenServers` provides the list of URLs to the server where the page is to be regenerated. If no URLs are specified, default value of PageCacheUpdater defaults to standard regeneration (based on user request).

RealTime Publishing uses asynchronous messaging to track the status of each job. It is not necessary to know the details of the messaging framework, but note that communication with the target system is facilitated through the Transporter. This also includes messages issued by Unpacker to inform the management system that an asset has been saved, prompting the management logic to mark that asset published. The publishing jobs per each target of the multi-target publish can complete in different orders because they are independently run on each target during the transport phase. For example, CacheFluster for target 1 might complete before Unpacker for target 2.

75.2 Writing a Custom Transporter

With a transporter, you can replace the HTTP(s)-based OOTB (out-of-the-box or ready to use) transport with another transport that uses a different protocol, and you can publish to multiple targets within the same publishing session.

Topics:

- Writing Your Own Transporter

- Considerations About Overriding AbstractTransporter Methods

- Helper Methods in AbstractTransporter

- Implementing a Transporter: Example

- Code for Writing RealTime Publishing Transporter

- Understanding Edge-Case Scenarios

- Intercepting Asset Publishing Events on the Management Instance

- Distinguishing Between Unpackers and CacheUpdates

75.2.1 Writing Your Own Transporter

Follow these steps to write your own transporter:

1. Subclass the `com.fatwire.realtime.AbstractTransporter` class.

2. Override the methods `ping`, `sendBatch`, `listTransports`, `toString`, and `remoteExecute`.

3. Install the transporter by editing the `classes/AdvPub.xml` file on the management side.

   Replace the line:

   ```
   <bean id="DataTransporter"
     class="com.fatwire.realtime.MirrorBasedTransporterImpl"
     singleton="false">
   ```

   with:

   ```
   <bean id="DataTransporter"
     class="[your transporter class]"
     singleton="false">
   ```

75.2.2 Considerations About Overriding AbstractTransporter Methods

When you override the `AbstractTransporter` methods, keep the following points in mind.

- `ping()` contains the logic that checks whether the target is up or down. Its most prominent use is to power the green/red diagnostic indicator in the publishing console. It is not necessary for ping to be successful to launch a publishing session, but this can be a handy tool for diagnosing connection problems.

  If you are using http(s) to connect to your target, you may be able to use the default implementation rather than override and implement your own.

- `sendBatch()` is responsible for uploading data to the remote `fw_PubDataStore`. It is invoked multiple times with small batches of data from the local `fw_PubDataStore` that comes in the form of an IList. Batching helps keep memory usage down and is done behind the scenes for you.

- `remoteExecute()` is responsible for communicating with the remote system. The communication is two-way: management sends commands to dispatch remote jobs and cancellation requests, while the target sends back messages that indicate its status. The contents of these messages are immaterial to `remoteExecute`, all it needs to do is send those requests and return the responses.

- `listTransports()` is a listing of the underlying transports, in case there are multiple targets. If there is only a single target, this method can just return a `toString()` rendition of the current transport.

- `toString()` is a human-friendly descriptor of this transport. For example, a typical value would be `http://mytarget:8081/cs/`. However, any other string is acceptable, including `targetDataCenter-Virginia`, `serverOn8080`, and so on.

75.2.3 Helper Methods in AbstractTransporter

A few helper methods are available in `AbstractTransporter`:

- `protected void writeLog(String msg)` writes messages to the publish log.

- `protected AbstractTransporter getStandardTransporterInstance()` get a new instance of the standard HTTP-based transporter. This can be useful to implement a transport to multiple targets.

- `protected String getParam(String param)` obtains the value of a publishing parameter, as configured in the publishing console.

75.2.4 Implementing a Transporter: Example

Following is an example of a transporter implementation that works with multiple targets. The target is configured as follows:

1. In the **Destination Address**, specify comma-separated destination URLs.

   For example:

   ```
   http://tgt1:9030/cs/
   http://virginia:9040/cs/
   ```

2. In the **More Arguments**, specify ampersand-separated user name, password, and optional proxy information for the additional servers, suffixed with indexes starting at 1.

For example, with one additional target:

```
REMOTEUSER1=fwadmin&REMOTEPASS1=xceladmin&PROXYSERVER1=proxy.com&PROXYPORT1=9090&P
ROXYUSER1=pxuser&PROXYPASSWORD1=pxpass
```

3. In `AdvPub.xml`, replace the `DataTransporter` bean entry with the following:

```xml
<bean id="DataTransporter"
  class="my.sample.MultiTransporter"
  singleton="false">
 <property name="id" value="Transporter"/>
</bean>
```

75.2.5 Code for Writing RealTime Publishing Transporter

```java
com.fatwire.realtime.mypackage;
import COM.FutureTense.Interfaces.*;
import com.fatwire.cs.core.realtime.TransporterReply;
import java.net.URL;
import java.util.*;
/**
 * RealTime Publishing transporter to multiple targets.
 */
public class MultiTransporter extends AbstractTransporter
{
  private boolean initialized = false;
  List<AbstractTransporter>  transporters  = new ArrayList();
  /**
   * Ping each underlying target and return true if all of them are up.
   */
  @Override
  public boolean ping(StringBuilder sbOut)
  {
  init();
  boolean ret = true;
  for(AbstractTransporter  t : transporters)
  {
  boolean thisret = t.ping(sbOut);
  sbOut.append(t.getRemoteUrl() + (thisret ? " OK" : " Not reachable"));
  sbOut.append(" ||| ");
  ret &= thisret;
  }
  return ret;
  }
  /**
   * Send the batch to each underliyng transport.
   */
  @Override
  protected int sendBatch(ICS ics, IList iList, StringBuffer outputMsg)
  {
  init();
  for(AbstractTransporter  t : transporters)
  {
  int res = t.sendBatch(ics, iList, outputMsg);
  if(res != 0)
  {
  // Just log the error for now, but this is an
  // indication that the target may be down
  // and other notifications may also be appropriate.
  writeLog("Transporter " + t + " failed with " + res   + " " + outputMsg);
  }
```

```
}
return 0;
}
/**
 * Execute the remote command on each transporter and
 * accumulate their responses.
 */
@Override
protected List<TransporterReply> remoteExecute(ICS ics, String s,
Map<String, String> stringStringMap)
{
init();
List<TransporterReply> res = new ArrayList<TransporterReply>();
for(AbstractTransporter  t : transporters)
{
List<TransporterReply> tres = t.remoteExecute(ics, s, stringStringMap);
res.addAll(tres);
}
return res;
}
/**
 * Do some initialization by parsing out the configuration
 * settings and instantiating a standard http transport
 * to each target.
 */
private void init()
{
if(!initialized)
{
String remoteURLs = getRemoteUrl();
int count = 0;
for(String remoteUrl : remoteURLs.split(","))
  {
  String suffix = (count == 0) ? "" : String.valueOf(count);
  AbstractTransporter t1 =
  AbstractTransporter.getStandardTransporterInstance();
  URL url;
  try
  {
    url = new  URL(remoteUrl);
  }
  catch(Exception e)
  {
    throw new RuntimeException(e);
  }
  t1.setRemoteUrl(remoteUrl);
  t1.setHost(url.getHost());
  t1.setUsername(getParam("REMOTEUSER" + suffix));
  t1.setPassword(getParam("REMOTEPASS" + suffix));
  t1.setUseHttps("https".equalsIgnoreCase(url.getProtocol()));
  t1.setContextPath(url.getPath());
  t1.setPort(url.getPort());
  t1.setProxyserver(getProxyserver());
  t1.setProxyport(getProxyport());
  t1.setProxyuser(getProxyuser());
  t1.setProxypassword(getProxypassword());
  t1.setHttpVersion(getHttpVersion());
  t1.setTargetIniFile(getTargetIniFile()); transporters.add(t1);
  ++count;
  }
initialized = true;
```

```
writeLog("Initialized transporters: " + toString());
}
}
/**
 * Provide a full listing of all underlying transports. This is
 * can be used by other components to determine
 * whether they need to perform special actions depending on
 * the number of targets. For example, asset publishing
 * status processing may need to buffer responses until they're
 * received from all targets before marking assets published.
 * @return
 */
@Override
public List<String> listTransports()
{
init();
List<String> list = new ArrayList();
for(AbstractTransporter  t : transporters)
{
list.add(t.toString());
}
 return list;
}
/**
 * Just a human-friendly description of the transport. This may  show
 * up in the  logs, so make it descriptive enougn.
 */
@Override
public String toString()
{
List<String> transs = listTransports();
StringBuilder sb = new StringBuilder();
for(String t : transs)
sb.append(t + " ");
return sb.toString();
}
}
```

75.2.6 Understanding Edge-Case Scenarios

While the example in Code for Writing RealTime Publishing Transporter works in the optimistic case where all targets are running, there will be times when one target has stopped for a shorter or longer period of time. If you only publish to one target but still mark assets as published, then the target that stopped is not synchronized. You can handle such scenarios in the following ways:

- If a target stops for a short period of time, you should not mark assets as published, but continue publishing to the target that is running. When the other target is restarted, you have all earlier assets still queued for publishing. Those assets are redundantly published to the first target as well, but over short periods of time this is a negligible overhead.

- If a target stays down for a long period of time, it may be best to remove it from the list of targets in the destination configuration (in this example, remove the second target from the Destination Address in the publishing configuration). That way, assets continue to be marked as published even though you have only one active target. When the second target is restarted, first perform a database and file system sync, and then add it back to the list of destination addresses.

75.2.7 Intercepting Asset Publishing Events on the Management Instance

In the first case above, you have to only mark assets as published after they are saved on all targets. To do so, implement custom notification logic as follows:

1. Extend `com.fatwire.realtime.messaging.AssetPublishCallback`.

2. Override the `notify()` and optionally the `progressUpdate()` method.

Sample Implementation for Steps 1 and 2

```
package my.sample;
import com.fatwire.assetapi.data.AssetId;
import java.util.HashMap;
import java.util.Map;
/**
 * Buffer asset save notifications until we've received one
 * from each target. Then mark asset published.
 */
public class AssetPublishCallbackMulti extends AssetPublishCallback
{
  Map<String, Integer> saveEventsCount  = new HashMap<String, Integer>();
  /**
   * Receive notifications about the asset status.
   * Currently the only available status is SAVED.
   */
  @Override
  public void notify(AssetId assetId, String status, String from)
  {
  String assetIdStr = String.valueOf(assetId);
  writeLog("Got " + status + " notification from "
  + from + " for " + assetIdStr);
  if("SAVED".equals(status))
    {
    Integer numNotifications;
    if((numNotifications = saveEventsCount.get(assetIdStr)) == null)
      {
      numNotifications = 0;
      }
    numNotifications = numNotifications + 1;
    saveEventsCount.put(assetIdStr, numNotifications);
    if(numNotifications == this.getTargets().size())
      {
      super.notify(assetId, status, from);
      writeLog("Marked " + assetIdStr  + " published");
      }
    }
  }
  /**
   * Intercept progress update messages. Can be used for
   * monitoring the health of the system but is not required.
   */
  @Override
  public void progressUpdate(String sessionId, String job,
  String where, String progress, String lastAction, char status)
  {
  super.progressUpdate(sessionId, job, where, progress, lastAction, status);
  }
}
```

3. Enable the callback in `AdvPub.xml` on the management side:

- Add `AssetCallback` bean.

- Register the bean with `PubsessionMonitor`.

Enabling the Callback Bean for Step 3

To add the `AssetCallback` bean:

```
<bean id="AssetCallback"
  class="my.sample.AssetPublishCallbackMulti"
  singleton="false"/>
```

To register the bean with `PubsessionMonitor`:

```
<bean id="PubsessionMonitor"
  class="com.fatwire.realtime.messaging.PubsessionMonitor"
  singleton="false">

  <constructor-arg index="0">
    <ref local="DataTransporter" />
  </constructor-arg>

  <constructor-arg index="1">
    <ref local="AssetCallback" />
  </constructor-arg>

  <property name="pollFreqMillis" value="5000" />
  <property name="timeoutMillis" value="100000" />
</bean>
```

75.2.8 Distinguishing Between Unpackers and CacheUpdates

When publishing to multiple destinations, it is useful to distinguish between their respective Unpackers and CacheUpdaters. This comes in handy when looking at the progress bars in the RT publishing console and looking at logs.

To make that distinction, simply edit the `AdvPub.xml` file on the target side, and change the ID values of the `DataUnpacker` and `PageCacheUpdater` beans.

For example:

```
<bean id="DataUnpacker"
  class="com.fatwire.realtime.ParallelUnpacker"
  singleton="false">
  <property name="id" value="Unpacker-Virginia2"/>
  ...
</bean>

<bean id="PageCacheUpdater"
  class="com.fatwire.realtime.regen.ParallelRegeneratorEh"
  singleton="false">
  <property name="id" value="CacheFlusher-Virginia2"/>
  ...
</bean>
```

76

Understanding Asset and Publish Events in WebCenter Sites

WebCenter Sites supports Asset events and Publishing events in the current release.

Topics:

- Asset Events
- Publishing Events

76.1 Asset Events

Asset events take place when assets are added, modified or deleted by a contributor or programmatically. Upon these events, the event framework looks up and executes the set of configured events.

This topic describes how to write and register an asset event listener.

- Writing an Asset Event Listener
- Registering an Asset Event Listener

76.1.1 Writing an Asset Event Listener

Asset listeners have to extend `AssetEventListener` to be notified of asset changes. A convenient base class `AbstractAssetEventListener` comes with WebCenter Sites. Extending from this class makes it easy to recognize the specific type of action that led to the event (add/modify/delete).

Implement a custom asset listener using the sample code. This code prints the asset IDs; however, you can also plug in custom business logic.

```
package com.mycompany
public final class CustomAssetEventListener extends AbstractAssetEventListener
{
    public void assetAdded(AssetId id)
    {
        System.out.println("Asset " + id + " added");
    }
    public void assetUpdated(AssetId id)
    {
        System.out.println("Asset " + id + " updated");
    }
    public void assetDeleted(AssetId id)
    {
        System.out.println("Asset " + id + " deleted");
    }
}
```

Blocking asset event listeners are invoked after the asset operation has taken place, but before committing the data. Non-blocking asset event listeners are invoked asynchronously.

76.1.2 Registering an Asset Event Listener

This section describes how to register an asset event listener.

> **Note:**
>
> WebCenter Sites ships with a standard listener that is used for search indexing. Do not alter or delete it.

Asset event listeners are registered in the `AssetListener_reg` database table.

Table 76-1 AssetListener_reg Database Table

ID (integer)	Unique Identifier for the Row
`listener(String)`	Fully qualified class name that implements `AssetEventListener`. For example: `com.mycompany.CustomAssetEventListener`
`blocking(Y or N)`	'Y' indicates that the listener is blocking (runs synchronously with the thread that generated the event). 'N' indicates that the listener is non-blocking (runs in a separate thread).

76.2 Publishing Events

Publishing events are events that the RealTime publishing framework generates at each step of the publishing process. This feature is intended to facilitate system monitoring (such as SNMP) and other housekeeping processes.

Topics:

- Writing a Publishing Event Listener

- Registering a Publishing Event Listener

76.2.1 Writing a Publishing Event Listener

Publishing listeners have to implement `PublishingEventListener`. A `PublishingEvent` passed into the listener indicates the specific event that caused the invocation.

Use the following example to implement a custom even listener that prints the `pubsession` ID to the console:

```
package com.mycompany;
public class CustomPublishingEventListener implements PublishingEventListener
{
    public void onEvent( PublishingEvent event ) throws EventException
    {
        System.out.println( "Publishing event fired for     pubsession: " +
event.getPubSessionId());
        System.out.println( "Publishing task : " + event.getTaskName());
```

```
        System.out.println( "Status of the task :  " + event.getStatus());
        System.out.println( "Message associated with the task : " +
event.getMessage() );
    }
}
```

Publishing consists of multiple tasks (data gathering, packaging, transport, and so on), and each of them generates events. The `PublishingEvent` class represents an event in the publishing task. An implementation can query the task and its status from the event, as shown above.

Each task generates events when the following states are reached:

- STARTED

- DONE

- CANCELLED

- SUBTASK_FINISHED

- FAILED

76.2.2 Registering a Publishing Event Listener

- Register Publish event listeners in the `FW_PublishingEventRegistry` database table.

Table 76-2   FW_PublishingEventRegistry Database Table

ID (integer)	Unique Identifier for the Row
`listener(String)`	Fully qualified class name that implements `PublishingEventListener`. For example: `com.mycompany.CustomPublishingEventListener`
`blocking(Y or N)`	'Y' indicates that the listener is blocking (runs synchronously with the thread that generated the event).
	'N' indicates that the listener is non-blocking (runs in a separate thread).

77

Customizing Content Audit Reports

You can customize content audit reports by adding custom charts to these reports or changing the OOTB charts.

Topics:

- [About the Content Audit Reports](#)

- [Customizing the Content Audit Report](#)

77.1 About the Content Audit Reports

The Content Audit report is a collection of charts that show statistics on visitor traffic for a given asset. A chart is a visualization component that enables users to view and analyze data on website visitors. These Chart components query the WebCenter Sites database for data that will be displayed in the Content Audit report.

To view the Content Audit report, access an asset's Inspect view and then click the **Report** icon in the asset's toolbar. By default, the report shows the following charts:

- Metrics Bar: Contains a set of tickers that display statistics on site visitor activity on the asset during the selected time interval.

- Authoring Statistics Chart: This report plots the number of assets created and edited against the number of assets published during the selected time period, excluding technical assets such as Template assets.

- Publish Details Report: This report provides details about publishing history and scheduled publishing jobs for the selected time period.

- Author Productivity Report: This report displays information about the content contributors who created and edited assets in the selected time period, on the given content management site.

- Content Tag Cloud: This report displays the most common tags that were applied to assets during the selected time range.

- Asset Type List: This report displays a list of the top five asset types on which users performed create and edit operations during the selected time range, on the given content management site.

- Top Internal Searches List: This report lists the top 10 search words and phrases that Contributor interface users entered into the **Search** field during the reported time range.

See Working with Content Audit Reports in *Oracle Fusion Middleware Using Oracle WebCenter Sites*.

You can customize the Content Audit report by creating and adding custom charts to the report, customizing the default charts shown on the report, and removing charts

from the report. You can either customize the default Content Audit report, or create your own Content Audit report which will override the default report.

77.2 Customizing the Content Audit Report

You can customize the Content Audit report by creating a chart and its rendering elements that you will use to implement the chart, and then adding the chart to the report.

Topics:

- Creating a Custom Chart for the Content Audit Report

- Adding a Custom Chart to a Report

77.2.1 Creating a Custom Chart for the Content Audit Report

To create a chart, you must first create a chart asset (using the `WCS_Chart` asset type) and then create its rendering elements under `CustomElements` in the ElementCatalog.

The following procedures use the example of a custom bar chart, that displays the number of page visits to a selected asset, to provide instructions for creating a custom chart:

- Create a Chart Asset

- Create Rendering Elements to Implement the Chart

> **Note:**
>
> For detailed information about rendering elements, see UI Controller. For information about storing custom elements, see Element Storage.

- Add the Chart to a Report

77.2.1.1 Create a Chart Asset

1. Log in to WebCenter Sites, select the site for which you want to create a chart, and then select the Admin interface.

2. In the navigation pane, select the Content bar, expand the **Insights** tree, and then expand the **Charts** node.

3. Under the **Charts** node, click **Add New**.

   The create form for the Chart asset opens.

4. Fill in the following fields:

   - **Name**: Enter a name for the chart (for example, Custom).

   - **Description**: Provide a description for the chart (for example, Custom bar chart).

5. Click **Save**.

77.2.1.2 Create Rendering Elements to Implement the Chart

To implement a custom chart, you must create the chart's rendering elements under `CustomElements`. The UI Controller then tests for these custom elements in three phases: Configuration, Action, and Presentation.

For this example, create the `<CustomChart>Action.jsp` and `<CustomChart>Html.jsp` elements for the chart under `CustomElements/avisports/UI/Layout/CenterPane/Insights/Charts`. This location specifies the chart to be shown on the Content Audit report in the avisports sample site.

> **Note:**
>
> The chart in this example does not require a configuration element (`<CustomChart>Config.jsp`) because there are no configuration settings to be processed by the UI Controller. See UI Controller.

1. Create the `<CustomChart>Action.jsp`. For example:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld" %>
<%@ taglib prefix="ics" uri="futuretense_cs/ics.tld" %>
<%@ taglib prefix="satellite" uri="futuretense_cs/satellite.tld" %>
<%@ taglib prefix="controller" uri="futuretense_cs/controller.tld"%>
<%@page import="java.util.*"%>
<%@page import="java.text.*"%>
<%@page import="org.codehaus.jackson.map.ObjectMapper"%>
<%@page import="com.fatwire.cs.ui.framework.UIException"%>
<%@page import="com.fatwire.ui.util.GenericUtil"%>
<cs:ftcs>
<controller:callelement elementname="UI/Layout/CenterPane/Insights/Charts/
DateRange">
</controller:callelement>
<%
try{

    // BUSINESS LOGIC TO RETRIEVE DATA FOR THE CHART
    //convert the results to json string and set in the request attribute.
    ObjectMapper m = new ObjectMapper();
      // CONVERT OBJECT TO JSON
    //String json = m.writeValueAsString(mainList);
    request.setAttribute("json", json);
} catch(UIException e) {
    request.setAttribute(UIException._UI_EXCEPTION_, e);
    throw e;
} catch(Exception e) {
    UIException uie = new UIException(e);
    request.setAttribute(UIException._UI_EXCEPTION_, uie);
    throw uie;
}
%>
</cs:ftcs>
```

2. Create the `CustomHtml.jsp`. For example:

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld" %>
<%@ taglib prefix="ics" uri="futuretense_cs/ics.tld" %>
```

```
<cs:ftcs>
<div id='<%=ics.GetVar("chartId")%>'>
<h3 style="text-align:center">Custom Page Visit Bar Chart</h3>
    <div data-bind="ojComponent: {
    component: 'ojChart',
    type: 'bar',
    selection: 'multiple',
    series: visitsValue,
    groups: timestampValue,
    timeAxisType: 'enabled',
    animationOnDisplay: 'auto',
    legend: {position: 'top'}
    }"
    style="width:100%;height:100%;">
    </div>
</div>

<script>
    require( ['ojs/ojcore', 'knockout', 'jquery', 'ojs/ojknockout', 'ojs/
ojcomponents', 'ojs/ojchart'],
            function(oj, ko, $)
            {
                function ChartModel() {
                    var data = <%=request.getAttribute("json")%>;

                    var visits = [];
                    for(var i=0; i<data.length; i++){
                        visits.push({name:data[i].name,
items:data[i].items});
                    }
                    var timestamps = (data.length > 0) ? data[0].timestamps : [];
                    this.visitsValue = ko.observableArray(visits);
                    this.timestampValue = ko.observableArray(timestamps);
                }
                var chartModel = new ChartModel();
                $(document).ready(
                    function()
                    {
                        ko.applyBindings(chartModel, document.getElementById('<
%=ics.GetVar("chartId")%>'));
                    }
                );
            }
    );
</script>
</cs:ftcs>
```

The element in this example plots the chart. This element creates a bar chart using Oracle Jet library and the data retrieved in `CustomAction.jsp`.

To view the chart in the Contributor interface, add it to the Content Audit report (`ReportHtml.jsp`).

77.2.1.3 Add the Chart to a Report

To view this chart in the Contributor interface, add it to the Content Audit report. See Adding a Custom Chart to a Report.

77.2.2 Modifying the Chart's Rendering Elements

1. In the ElementCatalog, copy the chart's rendering elements from `UI/Layout/CenterPane/Insights/Charts/<ChartName>` to `CustomElements`, under the same path, in the ElementCatalog. For example: `CustomElements/UI/Layout/CenterPane/Insights/Charts/<ChartName>`.

2. Open the rendering elements in a text editor and make your changes to the JSP files. See Custom Elements.

77.2.3 Adding a Custom Chart to a Report

1. Add the custom chart to the Content Audit report:

   - To add the chart to a custom Content Audit report, insert the following lines into the custom `ReportHtml.jsp`:

     ```
     <!-- Include the charts here -->
     <div class="block box event-box">
     <div id="PageVisitRankContainer" class="siteinsightschart"
     data-chartname="Custom"></div>
     </div>
     ```

   - To add the chart to the default Content Audit report, insert the following line into `UI\Layout\CenterPane\Insights\Dashboard\ContentAuditHtml.jsp`:

     > **Note:** Make a copy of this element to CustomElements before making the changes.

     ```
     <div class="siteinsightschart" data-chartname="<ChartName>"></div>
     ```

2. Verify the chart displays properly on the Content Audit report:

   a. Log in to WebCenter Sites, select the avisports sample site, and then select the **Contributor** icon.

   b. Open an Article or Page asset, and then click the **Reports** icon.

   The custom report loads with the custom chart:

Figure 77-1   Custom Content Audit Report with Custom Chart

78
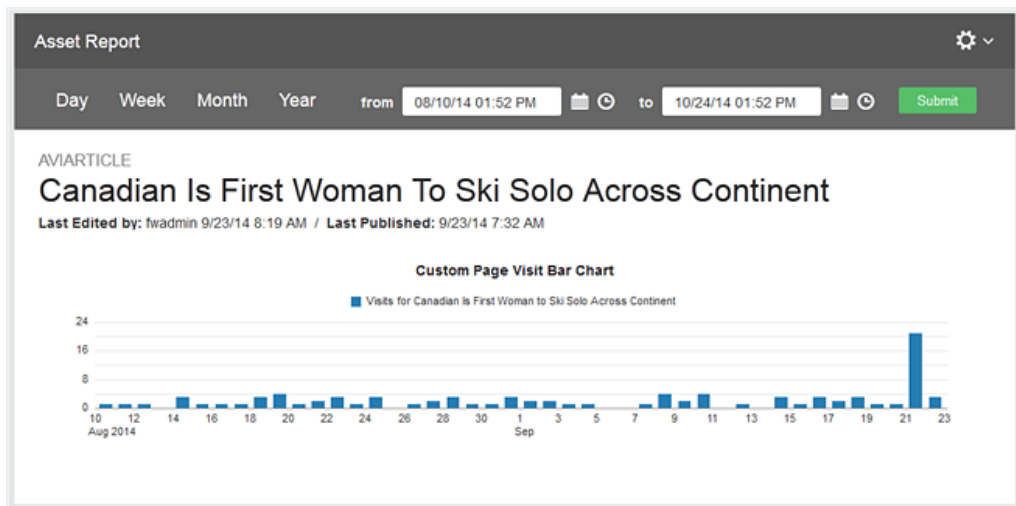
Adding Customizations to WebCenter Sites

WebCenter Sites 12c provides a modular and upgrade-safe way for you to add customizations using Weblogic Server's shared library mechanism. WebCenter Sites' runtime application refers this shared library, and therefore, customizations that you add to this shared library become available at runtime and in the WebCenter Sites' class path.

There are several benefits of using the shared library. When you upgrade WebCenter Sites, your customizations will remain intact, as they are externally managed in a separate shared library. Further, errors caused by customizations during deployment will be easier for you to debug.

Working with the Shared Library

WebCenter Sites runtime (`sites.war`) refers a shared library called extend.sites.webapp-lib.war. Out-of-the-box, this shared library is a placeholder. You must add all WebCenter Sites customizations to `extend.sites.webapp-lib.war`. Here are some examples of artifacts that you would share in the shared library:

- You are building an application that needs files such as `<customapp>.js`, an image (e.g. `<custom>.png`), and a CSS file (e.g. `<custom>.css`). You can add these files to `<custom>.war`.

- You are building a custom tag library that will be used for creating templates. This library includes `custom.tld` and `custom.jar`.

Here's how you create the shared library with your customizations:

1. Assemble custom resources (`.jar` file) and package a shared library (`.war` file)

2. Deploy the shared library to the managed server where WebCenter Sites is running

Assemble Custom Resources and Package Your Shared Library

1. Assemble the resources (images, css, etc) in a base directory, with the relevant subdirectories. See Assembling Shared Java EE Library Files in *Fusion Middleware Developing Applications for Oracle WebLogic Server* for more information about building and deploying shared libraries to the WebLogic Server.

2. Create a `META-INF/MANIFEST.MF` file describing the library:

```
Manifest-Version: 1.0
Specification-Title: Customizations
Specification-Version: 1.0
Implementation-Title: Custom Implementation
Implementation-Version: 1.0
Implementation-Vendor: Example.com
Extension-Name: images
```

3. Package the base directory as a WAR file. Here is what the packaging of the library looks like after you zip the library as a WAR file:

```
$ jar -tf customizations.war
META-INF/
META-INF/MANIFEST.MF
WEB-INF/weblogic.xml
mylogo.gif
WEB-INF/lib/custom.jar
WEB-INF/futuretense_cs/custom.tld
customapp.js
custom.png
```

Refer to the static resources from the library in the web application as if they are local to your application. If you plan to deploy two JARs together via a single shared library that your web application will reference, make sure that like the first library, the second `.jar` file is also placed in the `/WEB-INF/lib` folder.

> **Note:** The precedence of resource lookup is `sites.war` first and then the shared library. WebLogic Server serves the resource it finds first. That is:
>
> • Looks for the resource in the `sites.war` file.
>
> • If it doesn't find the resource in the `sites.war` file, it looks for it in the shared library.

Deploy Your Shared Library to the Managed Server Where WebCenter Sites is Running

1. Deploy the WAR as a library to the managed server where WebCenter Sites is running.

2. You can also deploy two JARs together via a single shared library that your web application will reference.

Part XIX

Troubleshooting

Get familiar with WebCenter Sites error logging and source code debugging system and general error logging and debugging techniques.

Topic:

- Logging and Debugging Errors

79

Logging and Debugging Errors

WebCenter Sites logs its activity in a log file, which in a new installation, is named `sites.log`, located in the `logs` folder. The type and volume of information that is written to the log file is controlled by the loggers that you choose to enable or define. WebCenter Sites also has a reserved variable that is used by JSP and XML tags for returning an error code if the tag did not successfully complete its task.

Topics:

- About Writing Custom Messages to the WebCenter Sites Log File

- Using Error Codes with Tags

79.1 About Writing Custom Messages to the WebCenter Sites Log File

You use the log ODL tool to view loggers and add new loggers. If you would like to write your own log messages to the WebCenter Sites log file, use the `ics:logmsg` tag.

WebCenter Sitesuses log ODL logging system. In log ODL, the `loggingconfig.xml` file specifies which information will be logged and how. The Admin interface provides the **Configure Log ODL** tool in the **System Tools** node, on the **Admin** tab. Using **Configure Log ODL**, you can configure log ODL and view loggers in the Admin interface. You can also dynamically add new loggers and change logger levels. Changes will persist upon system restart if you copy the text version of the loggers from the interface to the `loggingconfig.xml` file. See Using the Configure Log ODL Tool in *Oracle Fusion Middleware Administering Oracle WebCenter Sites*.

To define your own loggers or write your own messages to the WebCenter Sites log file, use the `ics:logmsg` tag. The following example writes a warning message to the WebCenter Sites log file.

```
<ics:logmsg msg="This is a warning message"
  name="com.fatwire.logging.cs.jsp" severity="warn"/>
```

For more information about `ics:logmsg`, see the *Oracle Fusion Middleware Tag Reference for Oracle WebCenter Sites Reference*.

> **Note:**
>
> It is recommended that you set loggers to a level that agrees with the type of system on which logging is implemented. On development and content management systems, logging levels can be set to a greater severity (such as `INFO` or `DEBUG`), which provides a large amount of information. On delivery systems, loggers can be either disabled or set to low severity (`WARN` or `ERROR`) to avoid performance setbacks and making system information available on a publicly accessed environment.

79.2 Using Error Codes with Tags

You can use a reserved variable named `Variables.errno` in WebCenter Sites when the JSP and XML tags don't successfully complete their task. Most JSP and XML tags use this variable for returning error codes (generally referred to as "errno").

For example, the `<CALLELEMENT>` XML tag sets `Variables.errno` as follows:

- `-10`: If you specified a nonexistent element.

- `-12`: If you specified an existing element that WebCenter Sites could not evaluate.

On success, `<CALLELEMENT>` does not modify the value of `Variables.errno`.

> **Note:**
>
> For revision tracking operations, the reserved variable named `Variable.errdetails` provides additional information about the error.

Use the following strategy with tags that use `Variables.errno`:

1. Initialize `Variables.errno` to `0` before calling the tag.

2. Call the tag.

3. Evaluate `Variables.errno`.

Tag Examples Using Error Codes

For example, the following code performs all three steps:

```
<SETVAR NAME="errno" VALUE="0"/>
<SETCOUNTER NAME="pi" VALUE="3.14159"/>
  <IF COND="Variables.errno=-501">
    <THEN>
      <p>Bad value of pi</p>
    </THEN>
  </IF>
```

Running this code yields the following HTML because `SETCOUNTER` cannot handle floating-point values:

```
<p>Bad value of pi</p>
```

The `ASSET`, `RENDER`, and `SITEPLAN` tags clear `errno` before they execute. You do not have to set `errno` to `0` when you use these tags. For example, after you use an `ASSET` tag, just check the value of `errno` to determine whether it has changed:

```
<ASSET.LOAD NAME="topArticle" TYPE="Article"
OBJECTID="Variables.cid"/>
  <IF COND="IsError.Variables.errno=false">
    <THEN>
      <ASSET.CHILDREN NAME="topArticle" LIST="listOfChildren"/>
    </THEN>
  </IF>
```

At the end of template elements, include error checking code such as this:

```
<IF rendermode="preview">
  <THEN>
```

```
    <IF COND="IsError.Variable.errno=true">
      <THEN>
        <FONT COLOR="#FF0000">
         Error <CSVAR NAME="Variables.errno"/>
         while rendering <CSVAR NAME="pagename"/>
         with asset ID <CSVAR NAME ="Variables.cid"/>.
        </FONT>
      </THEN>
    </IF>
  </THEN>
</IF>
```

Java Interface

After making calls to WebCenter Sites, the String variable `errno` can be retrieved and tested for success or failure. Here's an example:

```
cs.clearErrno();

IList rslt = cs.SelectTo(SYSTEMUSERS_TABLE, ALL_FIELDS, USERNAME,
            null, NO_LIMIT, null, CACHE_RESULTS, errstr);

errno = cs.GetVar("errno");

if (errno.compareTo(ERRNO_SUCCESS) == 0)
    {
     ...
```

Error Number Rules

Error numbers are always integers. This table summarizes error numbering rules for `Variables.errno`.

See the *Oracle Fusion Middleware Tag Reference for Oracle WebCenter Sites Reference* for specific error numbers for each tag.

Table 79-1   Error Number Rules

Number	Significance
Negative integers	Failure
0 (zero)	Success
Positive integers in a tag other than a revision tracking tag.	Information
Positive integers in a revision tracking tag.	Failure

Part XX

Reference

Learn about the Asset API, the search framework for managing search indices, and how you can use the search API to build public site searches.

Topics:

- Using Asset API: Tutorial

- Using Public Site Search

80

Using Asset API: Tutorial

Use this Asset API tutorial as a quick reference. Know that it's not a substitute for the *Oracle Fusion Middleware Java API Reference for Oracle WebCenter Sites*. Code samples in this tutorial will help you use the Asset API.

Topics:

- Understanding the Asset API

- Primary Interfaces

- Getting Started

- Asset API Read

- Asset API Write

- Development Strategies

- Optional: Setting Up to Use the Asset API from Standalone Java Programs

80.1 Understanding the Asset API

With the Asset API, you can access and manipulate WebCenter Sites assets. The main purpose of this API is to broaden the context in which you handle assets.

The Asset API is a Java API. Its major features are:

- The Asset API supports WebCenter Sites in a non-servlet context.

  Before the Asset API, WebCenter Sites exposed primary interfaces for programming in the form of tags (both in XML and JSP), used as the means for creating web pages and applications. In this sense, WebCenter Sites was tightly built around the servlet model and was not usable in other contexts, such as standalone Java programs (EJB, for example). Therefore, there was a need to create an API that could be used anywhere, regardless of the servlet framework.

- The Asset API unifies the retrieval, creation, and modification of the two asset families: basic and flex.

  - The Asset API represents both asset families with `AssetData` and `AttributeData`.

  - The Asset API uses generic `Condition` and `Query` objects for both flex and basic assets.

- The Asset API supports the creation and editing of basic assets, flex assets, and flex parent assets in the form of Java objects.

> **Note:**
>
> The Asset API does not log any dependencies other than the asset that is being loaded.

80.2 Primary Interfaces

With the Asset API, you can access data and definitions for WebCenter Sites assets.

The Asset API provides:

- Package `com.fatwire.assetapi.data` contains classes that are useful in reading data.

- Classes under `com.fatwire.assetapi.def` are for asset definitions.

- Package `com.fatwire.assetapi.query` contains constructs necessary for building a Query. See the *Oracle Fusion Middleware Java API Reference for Oracle WebCenter Sites*.

The Asset API defines the following primary interfaces:

- **Session**: The primary entry point into WebCenter Sites from the API. One has to obtain a session to be able to do anything at all in the Asset API.

- **AssetDataManager**: A manager for reading asset data. Developers can query for information here, and look up asset associations and other information.

- **AssetTypeDefManager**: A manager for reading an asset type's definition. 'Definition' is a loaded term in WebCenter Sites where flex assets are concerned. Here it is used in the generic sense, as something that defines the structure of an asset type. As a result, basic asset types also have a definition.

- **AssetData**: An asset's data; basically a collection of `AttributeData` instances and other information about the asset itself.

- **AssetId**: The asset type-ID combination.

- **DimensionableAssetManager**: A manager that supports multilingual assets by retrieving translations of any given asset.

80.3 Getting Started

The following are the prerequisites to follow this tutorial:

- FirstSiteII (FSII) is required to run the examples in this tutorial. The `tools.jar` file (available in JDK 1.5 and above) must be in the classpath.

  > **Note:**
  >
  > The Asset API can be used from a standalone Java program. Doing so requires some configuration. See Optional: Setting Up to Use the Asset API from Standalone Java Programs.

- Working through the examples in this chapter requires a knowledge of `jsp` elements and how they are created in the WebCenter Sites environment. See Creating Template, CSElement, and SiteEntry Assets.

80.4 Asset API Read

Let's start writing some code based on FSII data.

Topics:

80.4.1 A Simple Example: Reading Field Values

Let's try to read all the values of the **FSIIHeadline** field in FSII Articles. These are the steps to follow:

1. Get a session.

2. Get a handle to `AssetDataManager`.

3. Build a query.

4. Perform 'read' and print the results.

Here is the code that implements these steps (in a `jsp` element):

```
<%@ page import="com.fatwire.system.*"%>
<%@ page import="com.fatwire.assetapi.data.*"%>
<%@ page import="com.fatwire.assetapi.query.*"%>
<%@ page import="java.util.*"%>
<cs:ftcs>
<%
Session ses = SessionFactory.getSession();
AssetDataManager mgr =(AssetDataManager)
ses.getManager( AssetDataManager.class.getName() );
Query q = new SimpleQuery("Content_C", "FSII Article", null,
Collections.singletonList("FSIIHeadline") );
for( AssetData data : mgr.read( q ) )
{
out.println( data.getAttributeData("FSIIHeadline").getData() );
out.println( "<br/>" );
}
%>
</cs:ftcs>
```

1. `SessionFactory.newSession()` builds a session for a given user. From that point on, all data that is read using this session instance is based on the user's ACL permissions. You could also simply call `newSession( null, null )` and get a Session that belongs to `DefaultReader`, an assumed user. WebCenter Sites-powered web applications generally run as this user at runtime. However, error occurs if the incorrect user name and password are specified.

2. Using the session, get a handle to `AssetDataManager.getManager`. (The `AssetDataManager.class.getName()` method does this.)

3. A `Query` represents results that are based on the user's search criteria. In this example, we are using a simple version of `Query`, where we specify the asset type (Content_C) subtype (FSII Article) and the list of attributes to be returned (just `FSIIHeadline` in this case). We want all assets; therefore the third parameter (which takes `Condition` instance) is `null`. For information about how to use `Conditions`, see Running a Complex Query.

4. The `read()` method of `AssetDataManager` returns an `Iterable` over `AssetData`. Each piece of asset data contains an instance of `AttributeData` against an attribute name. `AttributeData.getData()` returns the real data of the attribute itself.

80.4.2 Reading AssetId

- To know the IDs of all these assets, use `AssetData.getAssetId()` which returns an `AssetId` instance.

- To print `AssetId`, modify the code in A Simple Example: Reading Field Values, as shown below:

```
for( AssetData data : mgr.read( q ) )
{
AssetId id = data.getAssetId();
out.println( data.getAttributeData("FSIIHeadline").getData() + " id=" + id );
out.println( "<br/>" );
}
```

The code above prints the following lines (note that `AssetId` is a composite that contains the id number and type. `AssetId.getId()` and `AssetId.getType()` return ID and type separately:

```
AudioCo. America Announces H300 series id=Content_C:1114083739888
AudioCo. New Portable Media Player Offers Full Video Experience id=Content_C:
1114083739926
AudioCo.'s First Under Water MP3 Player id=Content_C:1114083739951
...
```

80.4.3 Reading Attributes Given the Asset ID

You can also read the attributes of an asset ID (either passed into a template or acquired programmatically).

- Let's consider an `AssetId (Content_C:1114083739888`, for example, as shown in the code used in Reading AssetId) and attempt to print the name, description, and FSIIBody using the following code:

```
<%@ page import="com.fatwire.system.*"%>
<%@ page import="com.fatwire.assetapi.data.*"%>
<%@ page import="com.fatwire.assetapi.query.*"%>
```

```
<%@ page import="java.util.*"%>
<%@ page import="com.openmarket.xcelerate.asset.*"%>
<cs:ftcs>
<%
Session ses = SessionFactory.getSession();
AssetDataManager mgr =(AssetDataManager)
ses.getManager( AssetDataManager.class.getName() );
AssetId id = new AssetIdImpl( "Content_C", 1114083739888L );
List attrNames = new ArrayList();
attrNames.add( "name" );
attrNames.add( "description" );
attrNames.add( "FSIIBody" );

AssetData data = mgr.readAttributes( id, attrNames );
AttributeData attrDataName = data.getAttributeData( "name" );
AttributeData attrDataDescr = data.getAttributeData( "description" );
AttributeData attrDataBody = data.getAttributeData("FSIIBody");

out.println( "name:" + attrDataName.getData() );
out.println( "<br/>" );
out.println( "description:" + attrDataDescr.getData() );
out.println( "<br/>" );
out.println( "FSII Body:" + attrDataBody.getData() );
out.println( "<br/>" );
%>
</cs:ftcs>
```

Here, we are indicating which attributes to read for a given ID. As you can see, you can specify basic fields (such as name, description) and flex attributes; both are treated as attributes.

- Alternatively, you can load all attributes for a given ID by using `AssetDataManager.read( List<AssetId> ids)`. The following code demonstrates how this is done for a single `AssetId`:

```
<%@ page import="com.fatwire.system.*"%>
<%@ page import="com.fatwire.assetapi.data.*"%>
<%@ page import="com.fatwire.assetapi.query.*"%>
<%@ page import="java.util.*"%>
<%@ page import="com.openmarket.xcelerate.asset.*"%>
<cs:ftcs>
<%
Session ses = SessionFactory.getSession();
AssetDataManager mgr =(AssetDataManager)
ses.getManager( AssetDataManager.class.getName() );
AssetId id = new AssetIdImpl( "Content_C", 1114083739888L );

Iterable<AssetData> dataItr = mgr.read( Collections.singletonList( id ) );

for( AssetData data : dataItr )
{
for(AttributeData atrData : data.getAttributeData() )
{
out.println( "<br/>" );
out.println( "attribute name:" + atrData.getAttributeName() );
out.println( "data: " + atrData.getData() );
}
}
%>
</cs:ftcs>
```

80.4.4 Running a Query

A `Query` specifies the criteria based on which data is looked up.

- To look up a product whose SKU is `iAC-008` use the code below which also prints the name and ID.

```
<%@ page import="com.fatwire.system.*"%>
<%@ page import="com.fatwire.assetapi.data.*"%>
<%@ page import="com.fatwire.assetapi.query.*"%>
<%@ page import="java.util.*"%>
<cs:ftcs>
<%
Session ses = SessionFactory.getSession();
AssetDataManager mgr = (AssetDataManager)
ses.getManager( AssetDataManager.class.getName() );
Condition c = ConditionFactory.createCondition( "FSIISKU", OpTypeEnum.EQUALS,
"iAC-008" );
Query query = new SimpleQuery( "Product_C", "FSII Product", c,
Collections.singletonList( "name" ) );

for( AssetData data : mgr.read( query ) )
{
AttributeData attrData = data.getAttributeData( "name" );
out.println( "name:" + attrData.getData() );
out.println( "<br/>" );
out.println( "id:" + data.getAssetId() );
}
%>
</cs:ftcs>
```

Query consists of a `Condition`, a set of attributes to be returned, and a `SortOrder`. The example above uses a condition that is built on the `FSIISKU` attribute value being `EQUAL` to `iAC-008`. Just as we did in the previous example, we pass in the list of attribute names we want to be returned in the resulting collection of `AssetData`.

- There are some considerations as to what types of attributes can be queried and how. For a complete discussion of different query algorithms, see Query Types.

In short, some types of queries are possible with one algorithm, but not with the other. Note that this is exactly the behavior we have in the `Asset` family of tags and `AssetSet` family of tags. To illustrate this point, say we want to read all products whose price (`FSIIPrice`) is greater than 179.

`FSIIPrice` is of type `MONEY`. Consulting Table 80-2 in Data Types and Valid Query Operations, we see that the `GREATER_THAN` operation is allowed for this data type only in the basic/generic algorithm. The following code uses that algorithm to get all products whose price is greater than 179. The choice of query algorithm is made by the highlighted line in the code below:

```
<%@ page import="com.fatwire.system.*"%>
<%@ page import="com.fatwire.assetapi.data.*"%>
<%@ page import="com.fatwire.assetapi.query.*"%>
<%@ page import="java.util.*"%>

<cs:ftcs>
<%
Session ses = SessionFactory.getSession();
        AssetDataManager mgr = (AssetDataManager)
```

```
ses.getManager( AssetDataManager.class.getName() );
Condition c = ConditionFactory.createCondition( "FSIIPrice",
OpTypeEnum.GREATER_THAN, 179.0f );
Query query = new SimpleQuery( "Product_C", "FSII Product", c,
Arrays.asList( "name", "FSIIPrice" ) );
query.getProperties().setIsBasicSearch( true );

for( AssetData data : mgr.read( query ) )
{
AttributeData name = data.getAttributeData( "name" );
AttributeData price = data.getAttributeData( "FSIIPrice" );

out.println( "name:" + name.getData() );
out.println( "id:" + data.getAssetId() );
out.println( "price:" + price.getData() );

out.println( "<br/>" );
}
%>
</cs:ftcs>
```

80.4.5 Running a Complex Query

A complex query can be achieved through nested Conditions. According to the choice of query algorithm, the query is subject to the constraints listed in Query Types.

Use the following code to retrieve all the Product_C assets whose FSIIPrice attribute is greater than 179.0 or whose names are like "FSII" or both:

```
<%@ page import="com.fatwire.system.*"%>
<%@ page import="com.fatwire.assetapi.data.*"%>
<%@ page import="com.fatwire.assetapi.query.*"%>
<%@ page import="java.util.*"%>
<cs:ftcs>
<%
Session ses = SessionFactory.getSession();
AssetDataManager mgr = (AssetDataManager)
ses.getManager( AssetDataManager.class.getName() );
Condition c1 = ConditionFactory.createCondition( "FSIIPrice",
OpTypeEnum.GREATER_THAN, 179.0f );
Condition c2 = ConditionFactory.createCondition( "name", OpTypeEnum.LIKE, "FSII" );
Condition c = c1.and( c2 ); // c1.or( c2 );

Query query = new SimpleQuery( "Product_C", "FSII Product", c,
Arrays.asList( "name", "FSIIPrice" ) );
query.getProperties().setIsBasicSearch( true );

for( AssetData data : mgr.read( query ) )
{
AttributeData name = data.getAttributeData( "name" );
AttributeData price = data.getAttributeData( "FSIIPrice" );

out.println( "name:" + name.getData() );
out.println( "id:" + data.getAssetId() );
out.println( "price:" + price.getData() );

out.println( "<br/>" );
}
%>
</cs:ftcs>
```

80.4.6 Retrieving the Results by Sorting

You can retrieve the results by sorting on a field, and you can reverse the sort order, as follows:

- To retrieve the results sorted by price use the following code (specifying a SortOrder, ascending in this example):

```
<%@ page import="com.fatwire.system.*"%>
<%@ page import="com.fatwire.assetapi.data.*"%>
<%@ page import="com.fatwire.assetapi.query.*"%>
<%@ page import="java.util.*"%>
<cs:ftcs>
<%
Session ses = SessionFactory.getSession();
        AssetDataManager mgr = (AssetDataManager)
ses.getManager( AssetDataManager.class.getName() );

SortOrder so = new SortOrder( "FSIIPrice", true );
Query query = new SimpleQuery( "Product_C", "FSII Product",
null, Collections.singletonList( "FSIIPrice" ),
Collections.singletonList( so ) );

for( AssetData data : mgr.read( query ) )
{
AttributeData price = data.getAttributeData( "FSIIPrice" );

out.println( "id:" + data.getAssetId() );
out.println( "price:" + price.getData() );

out.println( "<br/>" );
}
%>
</cs:ftcs>

The above code sorts and prints asset ids and price in the ascending order of
FSIIPrice,

id:Product_C:1114083739851 price:89.99
id:Product_C:1114083739757 price:99.95
id:Product_C:1114083739696 price:129.99
id:Product_C:1114083739301 price:129.99
id:Product_C:1114083739471 price:179.95
id:Product_C:1114083739350 price:189.95
id:Product_C:1114083739225 price:399.99
id:Product_C:1114083739596 price:899.95
id:Product_C:1114083739804 price:1399.99
id:Product_C:1114083739549 price:3799.95
id:Product_C:1114083739663 price:6999.99
```

- To reverse the sort order, change true to false in the highlighted line of the code above.

80.4.7 Reading BlobObject

The Asset API defines a special class to represent file type of data, data that is stored as a binary file. For example, the FSIIDocumentFile attribute in FirstSiteII is of type blob.

- Use the following code to read `FSIIDocumentFile` from all `Document_C` instances and print the asset ID, file name, and file size:

```
<%@ page import="com.fatwire.system.*"%>
<%@ page import="com.fatwire.assetapi.data.*"%>
<%@ page import="com.fatwire.assetapi.query.*"%>
<%@ page import="java.util.*"%>
<cs:ftcs>
<%
Session ses = SessionFactory.getSession();
AssetDataManager mgr = (AssetDataManager)
ses.getManager( AssetDataManager.class.getName() );

Query query = new SimpleQuery( "Document_C", "FSII Document", null,
Collections.singletonList( "FSIIDocumentFile" ) );

for( AssetData data : mgr.read( query ) )
{
AttributeData docAttr = data.getAttributeData("FSIIDocumentFile");
BlobObject fileObj = (BlobObject)docAttr.getData();
byte [] d = new byte[fileObj.getBinaryStream().available()];
fileObj.getBinaryStream().read(d);

out.println( "id:" + data.getAssetId() );
out.println( "file name:" + fileObj.getFilename() );
out.println( "file size:" + d.length );

out.println( "<br/>" );
}
%>
</cs:ftcs>
```

80.4.8 Retrieving Multi-Valued Attributes

The Asset API supports multi-valued attributes in the same way it supports single-valued attributes. `AttributeData` contains a companion method, `getDataAsList()` to retrieve multiple values.

Use the following code to print data contained in `FSIIKeyword`, a multi-valued attribute:

> **Note:**
>
> Because sample data that ships with FirstSiteII does not have data for the `FSIIKeyword` attribute, this sample code does not print any keywords. Before running this code, edit some `FSIIDocument` instances to add data for this attribute.

```
<%@ page import="com.fatwire.system.*"%>
<%@ page import="com.fatwire.assetapi.data.*"%>
<%@ page import="com.fatwire.assetapi.query.*"%>
<%@ page import="java.util.*"%>
<cs:ftcs>
<%
Session ses = SessionFactory.getSession();
AssetDataManager mgr = (AssetDataManager)
ses.getManager( AssetDataManager.class.getName() );

Query query = new SimpleQuery( "Document_C", "FSII Document",
```

```
null, Collections.singletonList( "FSIIKeyword" ) );

for( AssetData data : mgr.read( query ) )
{
AttributeData attrData = data.getAttributeData( "FSIIKeyword" );
List retData = attrData.getDataAsList();
out.println( "id:" + data.getAssetId() );

for( Object o : retData )
{
out.println( "data:" + o );
}

out.println( "<br/>" );
}
%>
</cs:ftcs>
```

80.4.9 Multilingual Assets: Retrieving Translations

The Asset API also provides interfaces and methods to deal with multilingual assets. Basically, you have methods in `DimensionableAssetManager` to handle multilingual assets. They deal with getting all the locales for a given asset and specific translation of an asset.

Use the following example to first retrieve the translations for asset `Page: 1118867611403` by using the `getRelatives` method in `DimensionableAssetManager` with group set to `Locale`, and then use the `getRelative` method to get the `fr_FR` translation of the asset:

```
<%@ page import="com.fatwire.system.*"%>
<%@ page import="com.fatwire.assetapi.data.*"%>
<%@ page import="com.fatwire.mda.DimensionableAssetManager"%>
<%@ page import="com.openmarket.xcelerate.asset.*"%>
<%@ page import="java.util.*"%>

<cs:ftcs>
<%
Session ses = SessionFactory.getSession();
DimensionableAssetManager mgr =
(DimensionableAssetManager)ses.getManager(DimensionableAssetManager.class.getName());

AssetId page_asset = new AssetIdImpl("Page", 1118867611403L);

for( AssetId id : mgr.getRelatives( page_asset, null, "Locale" ))
{
out.println( id );
}
out.println( "<br/>" );

AssetId fr_translation = mgr.getRelative( page_asset, "fr_FR" );
out.println( fr_translation );

%>
</cs:ftcs>
```

80.4.10 Reading Asset and Attribute Definitions

In addition to asset data, APIs also provide access to their definitions. Information such as the attributes that make up an asset definition or the type of each attribute can be obtained through a manager called `AssetTypeDefManager`.

Use the following example to read all definition information from `Document_C` and print it to the browser:

```
<%@ page import="com.fatwire.system.*"%>
<%@ page import="com.fatwire.assetapi.def.*"%>
<%@ page import="java.util.*"%>
<%@ page import="com.fatwire.assetapi.query.*"%>
<cs:ftcs>
<%
Session ses = SessionFactory.getSession();
        AssetTypeDefManager mgr = (AssetTypeDefManager)
ses.getManager( AssetTypeDefManager.class.getName() );

AssetTypeDef defMgr = mgr.findByName( "Document_C", "FSII Document" );

out.println( "Asset type description: " + defMgr.getDescription() );
out.println( "<br/>" );

for( AttributeDef attrDef : defMgr.getAttributeDefs() )
{
out.println( "Attribute name: " + attrDef.getName() );
out.println( "Attribute description: " + attrDef.getDescription() );
out.println( "is required: " + attrDef.isDataMandatory() );
out.println( "Attribute type: " + attrDef.getType() );
out.println( "<br/>" );
}

%>
</cs:ftcs>
```

80.4.11 Reading Key-Value Mappings

The Asset API provides access to a given CSElement or template's key-value mapping pairs.

Use the following example to read all mapping pairs from a CSElement:

```
<%@ page import="com.fatwire.system.*"%>
<%@ page import="com.fatwire.assetapi.data.*"%>
<%@ page import="com.fatwire.assetapi.query.*"%>
<%@ page import="java.util.*"%>
<cs:ftcs>
<%
Session ses = SessionFactory.getSession();
AssetDataManager mgr = (AssetDataManager)
(ses.getManager(AssetDataManager.class.getName()));
Condition c = ConditionFactory.createCondition("name", OpTypeEnum.LIKE,
"FSIICommon/Nav/LocaleForm");
Query query = new SimpleQuery("CSElement", null, c, null);
query.getProperties().setReadAll(true);
for (AssetData data : mgr.read(query))
{
   List<AttributeData>  mappingArray =
(List<AttributeData>)data.getAttributeData("Mapping"). getData();
   for (int i=0; i<mappingArray.size(); i++)
```

```
      {
         HashMap mappingMap = (HashMap)mappingArray.get(i).getData();
         String key = (String)((AttributeData)mappingMap.get("key")).getData();
         String type =(String)
((AttributeData)mappingMap.get("type")).getData();
         String value = (String)((AttributeData)mappingMap.get("value")).getData();
         String siteid = (String)
((AttributeData)mappingMap.get("siteid")).getData();

         out.println("Mapping Entry #"+String.valueOf(i+1));
         out.println("<br/>");
         out.println("Key: "+key);
         out.println("Type: "+type);

out.println("Value: "+value);

         out.println("Siteid: "+siteid);
         out.println("<br/>");
   }
}
%>
</cs:ftcs>
```

80.5 Asset API Write

With the Asset API, you can perform write operations on basic assets and selected types of flex assets. The write operations are asset creation, modification, and deletion. The status field of an asset gets updated according to the action you perform.

The supported assets are basic assets, flex assets, and flex parents. These asset types are not supported at present: Flex Parent Definition, Flex Asset Definition, Flex Filter, and Flex Attribute. An `UnsupportedOperationException` is thrown if any write operation (insert or update) is attempted on those asset types.

Topics:

- Creating New Assets

- Updating Existing Assets

- Deleting Existing Assets

- Multilingual Assets

80.5.1 Creating New Assets

Asset API uses the `AssetDataManager.insert( List<AssetData> data )` method to create a new asset in WebCenter Sites. The method takes in a list of `AssetData` and uses these `AssetData` to create assets in WebCenter Sites. If successful, the method populates the passed in `AssetData` with the IDs of the newly created assets.

Asset API is able to create a new flex asset by combining the `AssetDataManager.newAssetData` method and `insert` method. The `newAssetData` method returns an empty `AssetData` with all the `AttributeData` objects populated with `null` or empty List/Collection. The `flextemplateid` (for flex assets) or `flexgroupid` (for flex parents) is automatically populated if the correct subtype is specified.

- To create a new flex asset, use the following code:

```
<%@ page import="com.fatwire.system.*"%>
<%@ page import="com.fatwire.assetapi.data.*"%>
<%@ page import="com.fatwire.assetapi.query.*"%>
<%@ page import="com.openmarket.xcelerate.asset.AssetIdImpl"%>
<%@ page import="java.util.*"%>
<cs:ftcs>
<%
Session ses = SessionFactory.getSession();
AssetDataManager adm = (AssetDataManager)
ses.getManager( AssetDataManager.class.getName() );
        MutableAssetData d = adm.newAssetData( "Content_C", "FSII Article" );
        d.getAttributeData( "name" ).setData( New Content" );
        d.getAttributeData( "FSIIHeadline" ).setData( "headline" );
        d.getAttributeData( "FSIIAbstract" ).setData( "abstract" );
        d.getAttributeData( "FSIIBody" ).setData( "body" );

d.getAttributeData( "Publist" ).setData( Arrays.asList( "FirstSiteII" ) );
        d.setParents( Arrays.<AssetId>asList( new AssetIdImpl( "Content_P",
1112192431478L)) );
        adm.insert( Arrays.<AssetData>asList( d ));
        out.println( d.getAssetId() );
%>
</cs:ftcs>
```

- Asset API is also capable of creating new basic assets. To create a new basic asset, use the following code:

```
<%@ page import="com.fatwire.system.*"%>
<%@ page import="com.fatwire.assetapi.data.*"%>
<%@ page import="com.fatwire.assetapi.query.*"%>
<%@ page import="java.util.*"%>
<cs:ftcs>
<%
Session ses = SessionFactory.getSession();
AssetDataManager adm = (AssetDataManager)
ses.getManager( AssetDataManager.class.getName() );
        MutableAssetData d = adm.newAssetData( "HelloArticle", "" );
        d.getAttributeData( "name" ).setData( New Hello Article" );
        d.getAttributeData( "headline" ).setData( "headline" );
        d.getAttributeData( "byline" ).setData( "abstract" );
        d.getAttributeData( "category" ).setData( "g" );
        BlobObject b = new BlobObjectImpl( "filename.txt", null,
"body".getBytes() );
        d.getAttributeData( "urlbody" ).setData( b );

d.getAttributeData( "Publist" ).setData( Arrays.asList( "HelloAssetWorld" ) );
        adm.insert( Arrays.<AssetData>asList( d ));
        out.println( d.getAssetId() );
%>
</cs:ftcs>
```

- You can create a new Content_C asset of subtype FSIIArticle and set `myAPItestvanity.html` as its vanity URL with the FSII webroot and an HTTP response code of 200, as shown in this example:

```
<%@ page import="com.fatwire.system.*"%>
<%@ page import="com.fatwire.assetapi.data.*"%>
<%@ page import="com.fatwire.assetapi.query.*"%>
<%@ page import="com.openmarket.xcelerate.asset.AssetIdImpl"%>
<%@ page import="java.util.*"%>
<cs:ftcs>
<%
```

```
                 Session ses = SessionFactory.getSession();
                 AssetDataManager adm = (AssetDataManager) ses.getManager(
                 AssetDataManager.class.getName() );
                     MutableAssetData d = adm.newAssetData( "Content_C", "FSII Article" );
                     d.getAttributeData( "name" ).setData( "New Content" );
                     d.getAttributeData( "FSIIHeadline" ).setData( "headline" );
                     d.getAttributeData( "FSIIAbstract" ).setData( "abstract" );
                     d.getAttributeData( "FSIIBody" ).setData( "body" );
                     d.getAttributeData( "Publist" ).setData( Arrays.asList( "FirstSiteII"
             ) );
                     d.setParents( Arrays.<AssetId>asList( new AssetIdImpl( "Content_P",
                 1112192431478L)) );
             .
                     //Retrieve a AttributeData reference to the asset's Webreference
                 attribute containing a list of vanity urls for the asset
                     AttributeData webReferenceAttrData =
                 d.getAttributeData("Webreference");

                     //Create new Vanity URL for this asset.
                     // Specify the follwing:
                 //String webroot: the name of the webroot to use
                 //String url: the vanity url
                 //Integer httpstatus: the http response code
                 //Long patternid: id of WebReferencesPatterns entry. 0 for urls not created
                 from pattern.
                 //boolean flag: indicates whether this the default vanity url for this
                 asset
                 //String template: SiteCatalog path for Template
                 //String wrapper: SiteCatalog path for Wrapper
                     WebReference vanityURL = new WebReferenceImpl("FSII",
                 "myAPItestvanity.html", new Integer("200"), 0L,
                 true,"FirstSiteII/FSIILayout","FSIIWrapper");
             .
                     //Set tthe asset's WebReference data attribute with the updated list:

                 webReferenceAttrData.setDataAsList(Arrays.<WebReference>asList(vanityURL));

                     //Save the asset
                      adm.insert( Arrays.<AssetData>asList( d ));
                      out.println( d.getAssetId()+"<br/>");
             .
             %>
             </cs:ftcs>
```

80.5.2 Updating Existing Assets

Another operation that Asset API supports, in addition to insert, is update. Similar to
insert, update saves the data from AssetData into WebCenter Sites, but to an existing
asset. If the asset does not exist, then the update operation throws an exception.

To update existing assets, use the following code:

```
<%@ page import="com.openmarket.xcelerate.asset.AssetIdImpl"%>
<%@ page import="com.fatwire.system.*"%>
<%@ page import="com.fatwire.assetapi.data.*"%>
<%@ page import="com.fatwire.assetapi.query.*"%>
<%@ page import="java.util.*"%>
<cs:ftcs>
<%
Session ses = SessionFactory.getSession();
AssetDataManager adm = (AssetDataManager)
```

```
ses.getManager( AssetDataManager.class.getName() );
Iterable<AssetData> assets = adm.read( Arrays.<AssetId>asList( new
AssetIdImpl( "HelloArticle", 1238171255471L), new AssetIdImpl( "Content_C",
1238171254486L)) );
        List<AssetData> sAssets = new ArrayList<AssetData>();
        for ( AssetData a : assets )
        {
            sAssets.add( a );
            a.getAttributeData( "name" ).setData( "Changed Name" );
        }
        adm.update( sAssets );
%>
</cs:ftcs>
```

80.5.3 Deleting Existing Assets

AssetAPI also supports deletion of assets from WebCenter Sites.

Use the following code to delete assets from WebCenter Sites.

After removing all the references to the two assets, `adm.delete` can delete both assets from WebCenter Sites. An exception is thrown if an asset is referenced by other assets, or if the asset is invalid. The delete process stops when an exception is thrown.

```
<%@ page import="com.openmarket.xcelerate.asset.AssetIdImpl"%>
<%@ page import="com.fatwire.system.*"%>
<%@ page import="com.fatwire.assetapi.data.*"%>
<%@ page import="com.fatwire.assetapi.query.*"%>
<%@ page import="java.util.*"%>
<cs:ftcs>
<%
 Session ses = SessionFactory.getSession();
 AssetDataManager adm = (AssetDataManager)
ses.getManager( AssetDataManager.class.getName() );
 adm.delete( Arrays.<AssetId>asList( new AssetIdImpl( "HelloArticle",
1238171255471L), new AssetIdImpl( "Content_C", 1238171254486L)) );
%>
</cs:ftcs>
```

80.5.4 Multilingual Assets

Asset API supports the creation of multilingual assets. Creation of the assets requires a two-step process. First, the asset is created and saved. Next, locale information is added.

Use the following example to create a new `Content_C` asset and set the locale to `en_US`.

```
<%@ page import="com.fatwire.system.*"%>
<%@ page import="com.fatwire.assetapi.data.*"%>
<%@ page import="com.fatwire.mda.*"%>
<%@ page import="com.fatwire.mda.DimensionableAssetInstance.
                 DimensionParentRelationship"%>
<%@ page import="java.util.*"%>
<%@ page import="com.openmarket.xcelerate.asset.*"%>
<%@ page import="com.openmarket.xcelerate.common.*"%>
<cs:ftcs>

<%
Session ses = SessionFactory.getSession();

AssetDataManager adm = (AssetDataManager)
```

```
ses.getManager( AssetDataManager.class.getName() );
MutableAssetData d = adm.newAssetData( "Content_C", "FSII Article" );
d.getAttributeData( "name" ).setData( New Content" );
d.getAttributeData( "FSIIHeadline" ).setData( "headline" );
d.getAttributeData( "FSIIAbstract" ).setData( "abstract" );
d.getAttributeData( "FSIIBody" ).setData( "body" );
d.getAttributeData( "Publist" ).setData( Arrays.asList( "FirstSiteII" ) );
d.setParents( Arrays.<AssetId>asList( new AssetIdImpl( "Content_P",
1112192431478L)) );
adm.insert( Arrays.<AssetData>asList( d ));
DimensionManager dam =
(DimensionManager)ses.getManager(DimensionManager.class.getName());
Dimension dim = dam.loadDimension("en_US");
d.getAttributeData("Dimension") .setData(Arrays.asList(new Dimension[]{dim}));
DimensionParentRelationship dpr = new DimParentRelationshipImpl("Locale",
d.getAssetId());
d.getAttributeData("Dimension-parent").setData(Arrays.asList(new
DimensionParentRelationship[]{dpr}));
adm.update( Arrays.<AssetData>asList( d ));
%>
</cs:ftcs>
```

80.6 Development Strategies

In this topic you will learn about data types and attribute data (maps WebCenter Sites data types to Java types) and query types (compares types of queries, their usage, and supported operations). AttributeData includes information about the WebCenter Sites-specific data type and the actual data. Using the Asset API, you can perform generic/basic and flex queries. The query that you run may have restrictions on what type of operation you can perform for a given data type.

Topics:

- Data Types and Attribute Data

- Query Types

- Data Types and Valid Query Operations

80.6.1 Data Types and Attribute Data

AttributeData contains information about the data type (WebCenter Sites specific type) and the actual data. The types are defined by AttributeTypeEnum. AttributeData.getData() and AttributeData.getDataAsList() return data objects of a specific Java type.

This table maps WebCenter Sites types to their corresponding Java types.

Table 80-1   WebCenter Sites Data Types and Java Types

WebCenter Sites Data Type	Java Type
INT	Integer
FLOAT	Double
STRING	String
DATE	Date

Table 80-1   (Cont.) WebCenter Sites Data Types and Java Types

WebCenter Sites Data Type	Java Type
MONEY	Double
LONG	Long
LARGE_TEXT	String
ASSET	AssetId
BLOB	BlobObject

80.6.2 Query Types

Using the Asset API, you can perform two kinds of queries: generic/basic and flex.

There are two different algorithms, one using the generic asset infrastructure (generic/basic query), and the other using AssetSets and Search States (flex query). Note that it is possible to use the generic/basic query for flex assets and basic assets; flex query, however, works only for flex assets.

Each of these algorithms has its advantages and disadvantages. The Asset API seeks to unify the querying mechanism and eventually let the API user not be concerned about the choice of algorithm. However, at the present time as there is no equivalence between these algorithms, the user needs to specify if she wants to use a specific feature, offered by one of the two.

`QueryProperties.setIsBasicSearch( true )` sets the query algorithm to generic/basic search for this query. It is set to `false` by default. For basic assets, the setting does not matter.

Which type of query to choose? Very simply put, to look for basic attributes of a flex asset, use the basic. Otherwise use the default. This works for most queries one generally encounters. Things are a bit more subtle than that. Given below are other considerations for each type of query.

Basic/Generic Query

- Cannot search on flex attributes if you do not specify subtype.

- Cannot search on a flex attribute that is not in the flex definition.

- Cannot sort on a flex attribute.

- Case sensitivity is not guaranteed (depends on the database).

- Only the `AND` operation is allowed between different fields (`name=name1 AND description=descr1` is allowed, but `name=name1 AND name=name2` is not).

- Only `OR` is allowed for two conditions involving the same field name. The `OR` condition does not work on flex attributes.

Flex Query

- Cannot have basic attributes in the condition (`id`, `name`, `description`, and so on).

- Cannot sort by basic attributes.

- Flex query works without a subtype being specified. The search applies to data of all subtypes.

- Can use only the following operands in the condition: LIKE, EQUALS, BETWEEN, and RICHTEXT.

80.6.3 Data Types and Valid Query Operations

Depending on the type of query being performed, there are further restrictions on what type of operation is allowed for a given data type.

In general, a flex type query (which is the default for flex assets) allows only the following OpTypeEnums; LIKE, EQUALS, BETWEEN, and RICHTEXT. Note that these are the same operations available from AssetSet/SearchState tags.

To use other OpTypeEnums, you have to use basic/generic query (by setting QueryProperties.setIsBasicSearch(true)). Such a query, of course, has to adhere to the basic query rules above.

This table shows the allowed set of operations per data type (single-valued or multi-valued) for a basic/generic query.

Table 80-2   Allowed Set of Operations

Data Type	EQUALS	NOT_EQUALS	LIKE	GREATER	LESSTHAN	BETWEEN	RICHTEXT
INT	Y	Y	N	Y	Y	–	N
FLOAT	Y	Y	N	Y	Y	–	N
STRING	Y	Y	Y	Y	Y	–	N
DATE	Y	Y	N	Y	Y	–	N
MONEY	Y	Y	N	Y	Y	–	N
LONG	Y	Y	N	Y	Y	–	N
LARGE_TEXT	N	N	Y	N	N	–	N
ASSET	Y	Y	N	N	N	–	N
BLOB	N	N	N	N	N	–	N

The following table shows the allowed set of operations per data type (single-valued or multi-valued) for the flex type query.

Table 80-3   Allowed Set of Operations per Data Type

Data Type	EQUALS	NOT_EQUALS	LIKE	GREATER	LESS THAN	BETWEEN	RICHTEXT
INT	Y	–	N	–	–	Y	N
FLOAT	Y	–	N	–	–	Y	N
STRING	Y	–	Y	–	–	Y	N
DATE	Y	–	N	–	–	Y	N

Table 80-3   (Cont.) Allowed Set of Operations per Data Type

Data Type	EQUALS	NOT_EQ UALS	LIKE	GREATE R	LESS THAN	BETWE EN	RICHTE XT
MONEY	Y	–	N	–	–	Y	N
LONG	Y	–	N	–	–	Y	N
LARGE_TEXT	N	–	Y	–	–	N	Y
ASSET	Y	–	N	–	–	Y	N
BLOB	N	–	N	–	–	N	Y

80.7 Optional: Setting Up to Use the Asset API from Standalone Java Programs

You can use the Asset API from JSP templates and also from a standalone Java program. When you want to use this API from a Java program, set up a single database connection or a connection pool outside WebCenter Sites.

> **Note:**
>
> In the following steps, we assume that all components are local to your WebCenter Sites installation.

Before setting up a single database connection or pool:

1. Make sure the following files are in the classpath of your Java program:

    *   `javaee.jar` and `tools.jar` (both are available in JDK 1.5 and higher versions).

    *   `ServletRequest.properties`. This file can be copied from the `WEB-INF/classes` folder.

    *   `SSOConfig.xml`, given that your WebCenter Sites installation has WEM installed and single sign-on (in `wcs_properties.json`) is set to `true`. The `SSOConfig.xml` file can be copied from the `WEB-INF/classes` folder.

    *   `cs-cache.xml`, `ss-cache.xml`, `linked-cache.xml`, and `cas-cache.xml`.

    *   `SitesSecurityContext.xml`. This file can be copied from the `WEB-INF/classes` folder.

    *   All of the WebCenter Sites binary files (`jar` files in the `WEB-INF/lib` folder).

2. Continue with the steps in one of the following sections:

    *   To set up a single database connection:

    *   To set up a database connection pool:

To set up a single database connection:

1. Set the following system properties for your Java program:

```
cs.dburl=<JDBC_URL_to_connect_to_DB>
cs.dbdriver=<driverClass>
cs.dbuid=<dbUserName>
cs.dbpwd=<dbPassword>
```

2. Locate the WebCenter Sites installation folder and pass its name as a JVM argument:

```
-Dcs.installDir=<install_dir>
```

To set up a database connection pool:

1. Add the following `jar` files to the classpath: `commons-dbcp.jar` (which ships with WebCenter Sites) and `commons-pool.jar` (available from the Apache website).

2. Create the property file with the same name as your data source:

   Note the following:

   - The name of the data source is the value of the `cs.dsn` property (in `wcs_properties.json`).

   - The value of `cs.dbconnpicture` (also in `wcs_properties.json`) must refer to `cs.dsn` (the combination of `cs.dbconnpicture` and `cs.dsn` must yield a valid resource).

     For example, the combination of

     `cs.dsn= csDataSource` *and*

     `cs.dbconnpicture= java\:comp/env/$dsn`

     yields a valid resource:

     `java:/csDataSource`

     Therefore, you would name the property file `csDataSource.properties` (the value of `cs.dsn`).

   - When creating the property file, make sure to place it in the classpath of your Java program.

3. When the property file is created, add the following keys to the file:

```
driver=<driverClass>
url=<JDBC_URL_to_connect_to_DB>
maxconnections=<number_of_connections_to_pool>
user=<dbUserName>
password=<dbPassword>
```

4. Locate the WebCenter Sites installation folder and pass its name as a JVM argument:

```
-Dcs.installDir=<install_dir>
```

81

Using Public Site Search

WebCenter Sites includes a new framework for managing search indices. This framework forms the basis for searches in both the editorial interface and on the live site. That is, the visitors' side. Hence the name public site search.

Topics:

- About the Search Framework

- Index Types

- About Search API

- Advanced Configuration

81.1 About the Search Framework

The search framework consists of the Search API, special asset event listeners, and a polling system for queues. You use this framework in coordination with the Event Management and Queue Management frameworks. This topic focuses primarily on the search framework.

> **Note:**
>
> You can skip this topic if you are primarily interested in the usage of the search API.

The following figure shows how the search engine integration framework works with the rest of WebCenter Sites.

Figure 81-1   Search Engine Integration
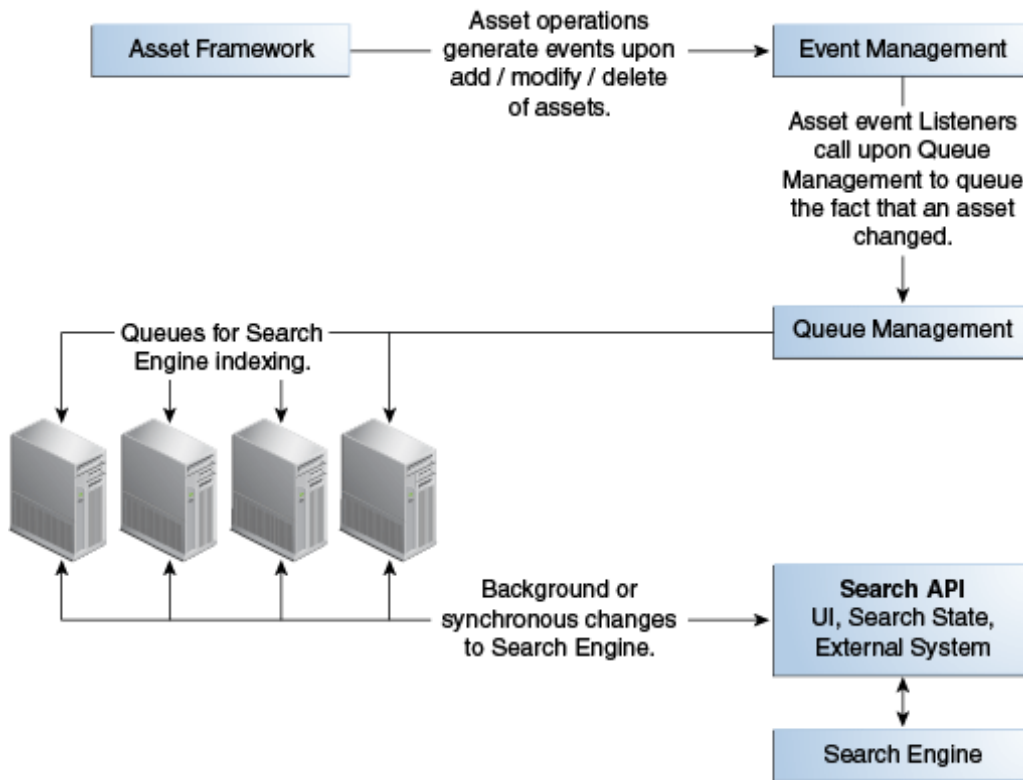
1. Asset framework detects changes/additions to assets and fires off events.

2. Registered listeners queue the changes, using a persistent queue implementation. A given event can be queued into one or many persistent queues. Each queue can be thought of as the source of data for a search index.

3. Once asset events are queued, a background process empties the queue contents and routes them to the Search API.

4. The Search API chooses the appropriate (configurable) search engine vendor implementation to start the indexing process.

81.2 Index Types

Two types of indices are created in WebCenter Sites: `Global` index and `AssetType` index. `Global` index is the index of all data (all asset types enabled for `Global` index). To search for a phrase or expression in multiple asset types (such as attempting to build a Google-like search interface), `Global` index is more appropriate.

While `Global` index contains data for all fields of the index, it does not store the data in a form that is suitable for parametric searches. An `AssetType` index contains indexed information for a given asset type in a manner that can be searched parametrically. The Admin interface supports the configuration of Asset Type searches, which includes attribute-based searches for the indexing-enabled asset types. See Adding Asset Types to the Search Index in *Oracle Fusion Middleware Administering Oracle WebCenter Sites*.

Topics:

- Global Index

- Asset Type Index

81.2.1 Global Index

`Global` index is used by the Oracle WebCenter Sites: Contributor interface to build a global search UI. An instance of the index also exists on the delivery server. The index on the delivery server can be used to build public site searches.

Only those assets that are published to the live site *after search is configured* are available for searches. It is during publishing that the data gets indexed. All assets that may exist on the live site before search is configured is not reflected in the `Global` index (until the assets are re-indexed on the live site).

A search index functions roughly similar to a database table.

The following table describes the fields in `Global` index.

> **Note:**
>
> The field names are case-sensitive.

Table 81-1   Fields in the Global Index

Field name	Description
defaultSearchField	This contains all the data of the indexed asset. This is the field you would search for in full-text searching.
	This contains index data for all attributes of the asset, including any binary field data. Data for all the attributes is merged into this single field and indexed. The index itself does not 'store' data for this field, but does allow full-text searching.
	Note: Search strings must be entered in lowercase only, no capitals.
id	This contains the asset ID.
AssetType	This contains the asset type (`Content_C/Product_P`).
locale	This contains the locale string (for example, `en_US`).
name	This contains the name of the asset.
description	Description associated with the asset.
subtype	Name of the subtype (flex definition name).
subtypeid	ID of the subtype (flex def ID).
updateddate	Last updated date as found at the time of indexing.
siteid	IDs of all sites in which this asset is available.
startdate	Start date field in the asset table.
enddate	End date field in the asset table.

81.2.2 Asset Type Index

An asset type index is created when it is enabled from the Admin interface by selecting the **Admin** tab, then **Search**, and then **Configure Asset Type Search**. Once an asset type is enabled, an index is created under /shared/lucene/<Asset type name>. This index contains all attributes of the given type as fields in the index.

The following table describes the fields in the Content_C index.

> **Note:**
>
> The field names are case-sensitive.

Table 81-2   Fields in the Asset Type Index

Field Name	Description
DefaultSearchField	This contains all the data of the indexed asset. This is the field you would search for in full text searching.
	This contains index data for all attributes of the asset, including any binary field data. Data for all the attributes is merged into this single field and indexed.
	The index itself does not 'store' data for this field, but does allow full text searching.
	Note: Search strings must be entered in lowercase only, no capitals.
id	This contains the asset ID
AssetType	This contains the asset type (Content_C/Product_P)
locale	Contains the locale string (example en_US)
name	Name of the asset
description	Description associated with the asset
subtype	Name of the subtype (flex definition name)
subtypeid	Id of the subtype (flex def ID)
updateddate	Last updated date as found at the time of indexing.
siteid	All site IDs this asset is available in
startdate	Startdate field in the asset table
enddate	Enddate field in the asset table
Dimension	ID of the dimension
Dimension-parent	ID of the Dimension parent
createdby	User name that created this asset
createddate	Date the asset was created

Table 81-2   (Cont.) Fields in the Asset Type Index

Field Name	Description
Publist	List of site names this asset belongs to
Relationships	Asset IDs of related items (flex only)
externaldoctype	Not used
filename	File name used for static publishing
flextemplateid	ID of the flex definition (flex only)
fw_uid	Globally unique ID of this asset
path	Path used for static publishing
renderid	Object ID of the Template asset assigned to a flex asset.
ruleset	XML document of the ruleset
status	Status associated with the asset
subtype	Subtype name
subtypeid	Subtype ID (flex only)
template	Template name
updatedby	User name that last updated this asset
updateddate	Date of last update
urlexternaldoc	Not used
urlexternaldocxml	Not used
FSIIAbstract	Flex attribute
FSIIBody	Flex attribute
FSIIByline	Flex attribute
FSIIDescriptionAttr	Flex attribute
FSIIHeadline	Flex attribute
FSIINameAttr	Flex attribute
FSIIPostDate	Flex attribute
FSIISubheadline	Flex attribute
FSIITemplateAttr	Flex attribute

To visualize which fields are available in a given index, use the tool Luke, available at:

http://www.getopt.org/luke/

After you launch the tool, use the tool's browse function to load the index by simply locating the folder that contains the index (for example: `../shared/lucene/ Content_C`).

81.3 About Search API

When you use the Search API, you'll work with the `SearchEngine` interface and the `QueryExpression` interface.

Topics:

- SearchEngine

- QueryExpression

- Configuring Query Expression

81.3.1 SearchEngine

The `SearchEngine` interface defines the key functions of a search engine implementation; indexing and searching. More information about SearchEngine is available in the *Oracle Fusion Middleware Java API Reference for Oracle WebCenter Sites*.

- The source of index data is given to `SearchEngine`. `SearchEngine`, in response to the indexing request, creates the search index (if it is not created), and updates the contents based on the `IndexSource` accessors.

- `index()` works off of a given `IndexSource` instance. Depending on the search engine's implementation details, it operates on new, modified, and deleted data coming from the `IndexSource` (in most search engines, all that is modified must be deleted first and then re-indexed).

    `index()` also invokes index lifecycle methods (`startIndexing()` and `endIndexing()`) on the given instance of `IndexSource`.

- `search()` operates on a `QueryExpression` against one or many indexes (`IndexSources`), resulting in a single set of results, sorted by their relevance (or `SortOrder`, if specified and usable across indices.

- A configuration lookup interface (`IndexSourceConfig`) is supplied to the `SearchEngine` instance which it can look up `IndexSource` properties, if needed.

- A `QueryConverter` interface is supplied to `SearchEngine`. This interface converts a given `QueryExpression` to its native form (recognizable by the specific search engine). This makes it possible to control the query language that the search engine uses externally.

- `SearchResult` is an abstraction over what is returned from the search engine. `SearchResult` is an iterator over `ResultRow`, a sub class of `IndexRow`, that contains relevance information. The `getRelavence()` method returns a double; the higher the value, the higher is the relevance of this `ResultRow` for the given query.

81.3.2 QueryExpression

The `QueryExpression` interface is a generic interface for defining search criteria. All search engines support native formats for building queries. The native form contains

definitions of wildcards, relevance hints, and so on. These tend to be very specific to each search engine.

Search engines also provide a basic query construct, which can be programmatically built (`AND` & `OR` over field matches). These can be thought of in terms of generalized programmable interfaces, although limited in power.

`QueryExpression` encapsulates four distinct characteristics of search engine queries:

- Native text search format: Most search engines support a very sophisticated native format for search, including wild cards, special hints, and so on. This is available through `getStringFormat()`.

- Conjunction and disjunction: ANDs and ORs of conditions using `and()` and `or()` methods.

- Pagination: Using `getStartIndex()` and `getMaxResults()` methods.

- Sorting: using `getSortOrder()`.

81.3.3 Configuring Query Expression

1. Ensure search indexing is enabled:

   In the `SystemEvents` table, verify that the `SearchIndexEvent` is enabled (`enabled field =1`). This is configured to run in the background constantly (`*:*:* */*/*`); in practice it runs about every 30 seconds.

2. Make sure Asset listener is registered:

   Assets are queued for processing by the search framework, using asset events. The asset events are registered in the `AssetListener_reg` table. Make sure the entry in the following table exists. Add it if it doesn't exist.

Table 81-3   AssetListener_reg Table

ID	Listener	Blocking
1153937286234	`com.openmarket.basic.event.SearchAssetIdE ventListener`	Y

`IndexSourceMetaDataConfig`: table that stores configuration information for `IndexSource`. This describes the structure and nature of the index itself. This should have a row for `Global` by default. Any asset type enabled for Asset type index has an additional row in this table.

`SearchEngineMetaDataConfig`: stores the search engine configuration. This table should have a row for Lucene, by default.

These are configured correctly by the installer and managed by the Admin interface.

Defaults here should suffice. See Advanced Configuration.

81.4 Advanced Configuration

While in most cases you will find that the defaults are sufficient, in some use cases you may need to configure Lucene parameters and AnalyzerFactory.

Topics:

- [Configuration of Lucene Parameters](#)
- [Configuration of Custom AnalyzerFactory](#)

81.4.1 Configuration of Lucene Parameters

> **Note:**
>
> Some parameters can cause significant changes in the way the index performs at run time. Refer to the Lucene documentation and rely on experimentation to determine the best settings for your site. It is highly advised that you keep the defaults unless you have compelling reasons to change them.

In the Lucene search engine, an index can be created with a certain set of parameters that determine how the index is created and how it performs. While the Lucene default parameters are reasonable, WebCenter Sites provides administrators with a way to change them.

The `SearchEngineMetaDataConfig` table contains one row per index. Each row has a field named `properties` whose contents are used to configure Lucene parameters. Parameter-value pairs are separated by a semicolon (';') as shown below:

```
param1=value1;param2=value2
```

This table describes the parameters supported by WebCenter Sites.

Table 81-4   Parameters Supported by WebCenter Sites

Parameter	Type	Description
mergeFactor	Integer	Determines how often segment indices are merged.
		With smaller values, less RAM is used while indexing, and searches on unoptimized indices are faster, but indexing speed is slower.
		With larger values, more RAM is used during indexing, and while searches on unoptimized indices are slower, indexing is faster. Thus larger values (> 10) are best for batch index creation, and smaller values (< 10) for indices that are interactively maintained.
		This must never be less than 2. The default value is 10.
maxMergeDocs	Integer	Determines the largest number of documents ever merged. Small values (for example, less than 10,000) are best for interactive indexing, as this limits the length of pauses while indexing to a few seconds. Larger values are best for batched indexing and speedier searches.
		Defaults to max integer value ($2^{31}-1$).

Table 81-4   (Cont.) Parameters Supported by WebCenter Sites

Parameter	Type	Description
maxBufferedDocs	Integer	Determines the minimal number of documents required before the buffered in-memory documents are merging and a new Segment is created. Defaults to 10.
optimizeInterval	Integer	Determines the time interval (in seconds) between optimize() calls. The default value is 30 seconds, which is the recommended value for most systems. To allow a large amount of data changes, set this parameter to any value within the range of 300 to 600 seconds.
commitLockTimeout	Long	Sets the maximum time to wait for a commit lock (in milliseconds). Defaults to 10000.
maxFieldLength	Integer	Maximum number of terms that are indexed for a single field in a document. This limits the amount of memory required for indexing, so that collections with very large files will not crash the indexing process by running out of memory. Note that this effectively truncates large documents, excluding from the index terms that occur further in the document. To support large source documents, be sure to set this value high enough to accommodate the expected size. If you set it to max value of Integer (231-1), then the only limit is memory, but you should anticipate an OutOfMemoryError. By default, no more than 10,000 terms will be indexed for a field.

Table 81-4   (Cont.) Parameters Supported by WebCenter Sites

Parameter	Type	Description
termIndexInterval	Integer	Sets the interval between indexed terms. Large values cause less memory to be used by IndexReader, but slow random-access to terms. Small values cause more memory to be used by an IndexReader, and speed random-access to terms.
		This parameter determines the amount of computation required per query term, regardless of the number of documents that contain that term. In particular, it is the maximum number of other terms that must be scanned before a term is located and its frequency and position information may be processed.
		In a large index with user-entered query terms, query processing time is likely to be dominated not by term lookup but rather by the processing of frequency and positional data. In a small index or when many uncommon query terms are generated (for example, by wildcard queries) term lookup may become a dominant cost. In particular, numUniqueTerms/interval terms are read into memory by an IndexReader, and, on average, interval/2terms must be scanned for each random term access.
		Default value is 128.
useCompoundFile	String (must be yes or no)	Setting to turn on usage of a compound file. When on, multiple files for each segment are merged into a single file once the segment creation is finished.
writeLockTimeout	Long	Sets the maximum time to wait for a write lock.
		Default value is 1000.

81.4.2 Configuration of Custom AnalyzerFactory

In Lucene, an Analyzer represents a policy for extracting index terms from text. Analyzers are used at the time of indexing and searching for various tasks such as removing stop words and removing white spaces.

Different Analyzers exist in the Lucene repository for handling various locales. Often Analyzers are used for injecting synonyms or addressing accented characters gracefully. You can also build your own Analyzer by using any of the Lucene standard analyzers as a basis.

The WebCenter Sites Lucene implementation uses StandadAnalyzer, a general purpose analyzer for the English language. However, WebCenter Sites supports custom Analyzers through a plugin interface, AnalyzerFactory. The configured AnalyzerFactory is used to look up the analyzer, when required, in the process of indexing or searching. The AnalyzerFactory looks up the analyzer in the following instances:

- When building the index as a whole

- When parsing a query

- When indexing an individual row

To plug in a custom `AnalyzerFactory`, you have to implement and register the `AnalyzerFactory` interface. Registration is done by modifying a row in the `SearchEngineMetaDataConfig` table. Add the following to the **properties** field of the row whose **Name** field is set to `Lucene`.

```
AnalyzerFactory=<fully qualified class name of your custom AnalyzerFactory>
```

Part XXI

Appendixes for Oracle WebCenter Sites Core

There are WebCenter Sites tools and utilities that you can use with the WebCenter Sites browser-based interface for developing and maintaining your websites. You can compress undesirable white space to reduce the size of the response as well as consumption of bandwidth. You can use WebCenter Sites URL assemblers to manage URL assembly and disassembly and define the appearance of URLs.

Topics:

- Introducing WebCenter Sites Tools and Utilities

- Understanding White Space and Compression

- Using WebCenter Sites URL Assemblers

82

Introducing WebCenter Sites Tools and Utilities

For developing and maintaining your websites, you use tools and utilities such as WebCenter Sites Explorer Tool, CatalogMover, XMLPost Property Management Tool, together with the WebCenter Sites browser-based interface.

Topics:

- Oracle WebCenter Sites Explorer
- Connecting to a WebCenter Sites Database
- CatalogMover
- Property Management Tool
- About Importing with XMLPost

82.1 Oracle WebCenter Sites Explorer

Oracle WebCenter Sites Explorer tool, a Microsoft Windows application, for viewing and editing tables and rows in the WebCenter Sites database, and for creating and editing executable elements (or files) written in XML or JSP.

You use Oracle WebCenter Sites Explorer to do the following:

- Add entries to tables
- Edit rows within tables
- Track revisions to rows of tables
- Create and drop WebCenter Sites tables
- Organize tables and folders into projects
- Preview `SiteCatalog` records as pages in a browser
- Export and import records as integrated `.cse` type files
- Export and import tables and projects in `.zip` files

Oracle WebCenter Sites Explorer is installed along with WebCenter Sites.

82.2 Connecting to a WebCenter Sites Database

You can use Oracle WebCenter Sites Explorer on any remote Microsoft Windows computer simply by copying the Oracle WebCenter Sites Explorer directory from a computer on which WebCenter Sites is installed (`<domain home>/wcsites/`

clients/oracle.wcsites.explorer.zip) to a directory on the remote computer.

You then unzip oracle.wcsites.explorer.zip and start the Oracle WebCenter Sites Explorer executable file (ContentServerExplorer.exe). Log in to WebCenter Sites by supplying a user name, password, host name, port, and protocol information.

To connect to a system that is running WebCenter Sites:

1. Start Oracle WebCenter Sites Explorer.

2. Choose **File** then **Open WebCenter Sites** to display the Login dialog.

3. Enter the following values:

   Name: Your WebCenter Sites user name.

   Password: Your WebCenter Sites password. (Depending on your site security, it may not be necessary to enter a name and password.)

   Host Name: The host name or IP address. You cannot leave this field blank.

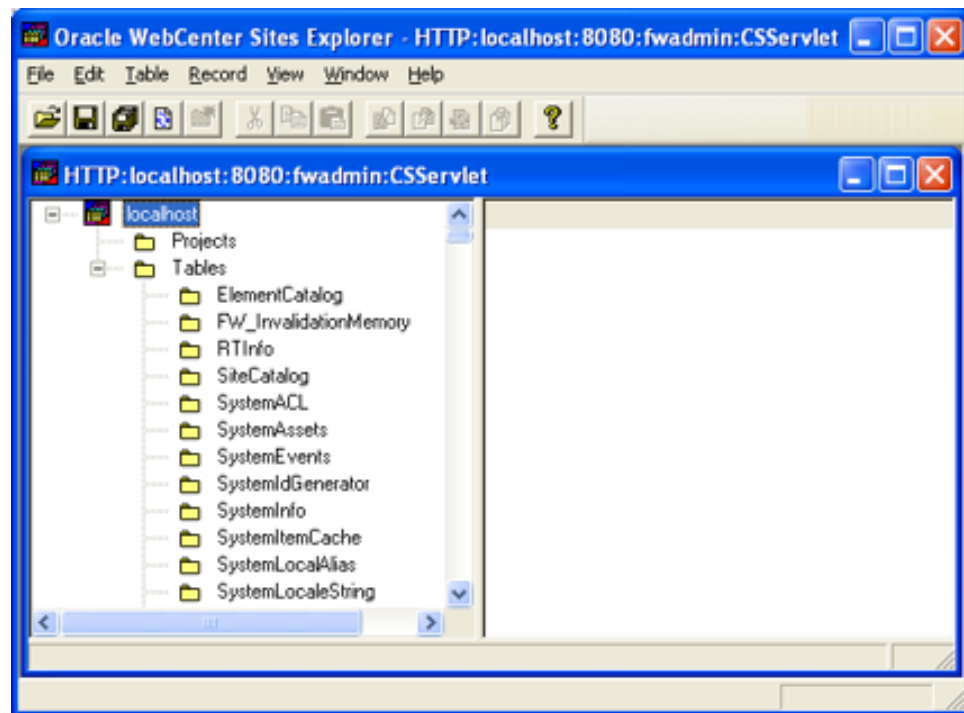   **Port**: The port number (the default is 80).

   Protocol: Typically, this is HTTP. You may select HTTPS if the web server is running SSL.

   Application Server URL Path: The type of application server for your site.

4. Click **OK** to log in.

   The Oracle WebCenter Sites Explorer utility opens.

Figure 82-1 Oracle WebCenter Sites Explorer

You may want to create a shortcut on your Windows desktop to Oracle WebCenter Sites Explorer. For instructions about using Oracle WebCenter Sites Explorer, see the

Oracle WebCenter Sites Explorer online help and sections in this guide that describe specific tasks requiring Oracle WebCenter Sites Explorer.

82.3 CatalogMover

You use the CatalogMover tool to export and import WebCenter Sites database tables, including the `ElementCatalog` and `SiteCatalog` tables. You can export and import database tables as either HTML files or ZIP files.

You can use CatalogMover through either the Windows interface described in the following sections, or the command line interface described in Command Line Interface.

> **Note:**
>
> In previous versions of WebCenter Sites, tables in the WebCenter Sites database were called *catalogs*. This term still applies to the names of some database tables and to the CatalogMover tool itself.

Topics:
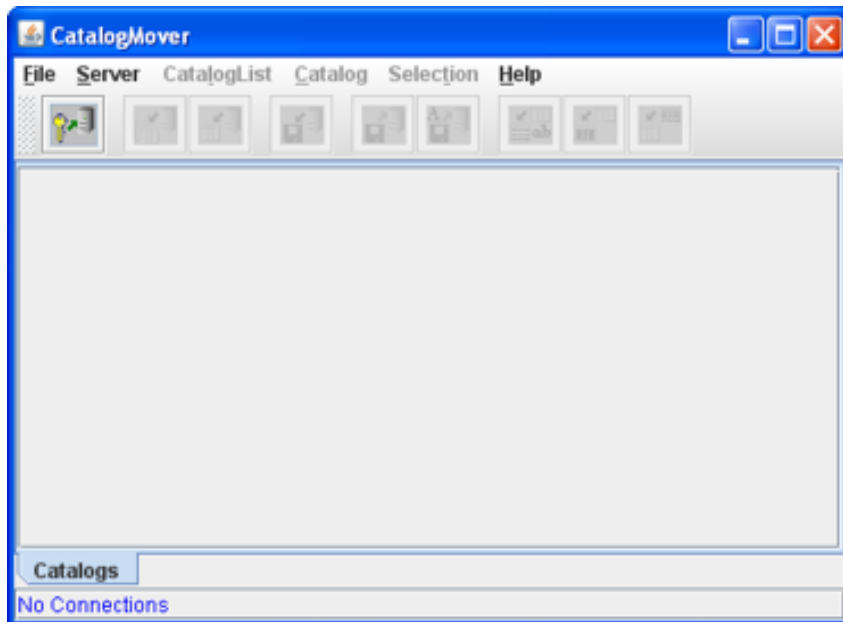
- Starting CatalogMover
- Connecting to WebCenter Sites
- CatalogMover Menu Commands
- Catalog Menu
- Exporting Tables
- Importing Tables
- Command Line Interface

82.3.1 Starting CatalogMover

To start CatalogMover, run the following scripts at the MS DOS prompt or in a UNIX shell:

- Windows: `catalogmover.bat`
- Solaris: `catalogmover.sh`

  The CatalogMover utility opens.

Figure 82-2   CatalogMover

82.3.2 Connecting to WebCenter Sites

Before using CatalogMover, you must first connect to a WebCenter Sites system.

1. To connect to WebCenter Sites, choose **Server** then **Connect**.

   The Connect to Server dialog opens.

   Figure 82-3   Connect to Server Dialog
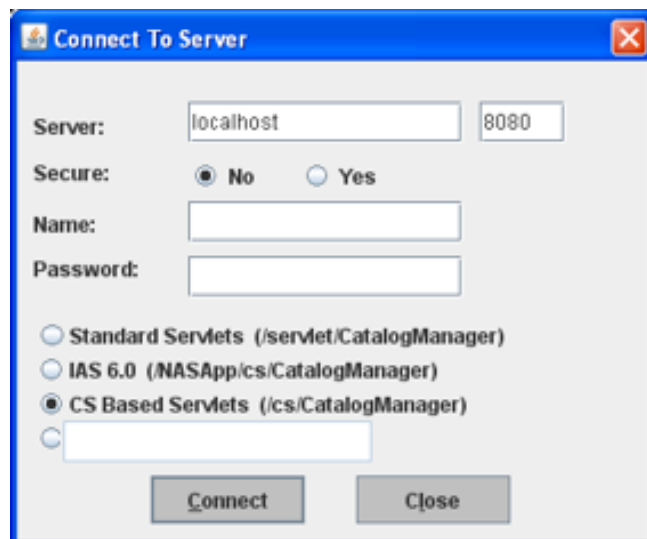
2. In the **Server** field, enter the name of the HTTP server you want to connect to.

3. For the **Secure** option, select **No** (default port 80) or **Yes** (default port 443). In the port field, enter the port (if not default) on which the server is running.

4. In the **Name** field, enter your user name.

5. In the **Password** field, enter your password.

6. Select one of the following options:

   - **Standard Servlets**: To connect to a system using WebSphere or WebLogic.

   - **IAS 6.0**: To connect to a NAS-App system.

   - **CS Based Servlets**: To connect to the CS-based servlets.

   - **Custom**: To connect to a different application server, enter the following value in the field (referencing the `.sh` or `.bat` script):

     `<ft.approot><ft.cgipath>/catalogmanager.`

7. Click **Connect**.

82.3.3 CatalogMover Menu Commands

CatalogMover includes the following menu commands:

File Menu

- **Exit**: Disconnect from WebCenter Sites and close CatalogMover.

Server Menu

- **Connect**: Display the Connect to Server dialog.

- **Reconnect**: Display the Connect to Server dialog and renew the current WebCenter Sites connection.

- **Disconnect**: Disconnect from WebCenter Sites.

- **Purge Temporary Tables**: Purge imported tables before committing.

- **Commit Individual Tables**: Commit imported tables to the database.

- **Normalize Filenames on Export**: Enable CatalogMover's file name normalization behavior, which changes the names of files that are being moved to names that match their corresponding ID numbers. File names are not altered if this feature is not enabled.

CatalogList Menu

- **Load**: Display a list of all tables in the database.

82.3.4 Catalog Menu

- **Load**: Load into local memory a table from the list.

  This figure shows a loaded `ElementCatalog` table.

Figure 82-4   ElementCatalog Table in CatalogMover
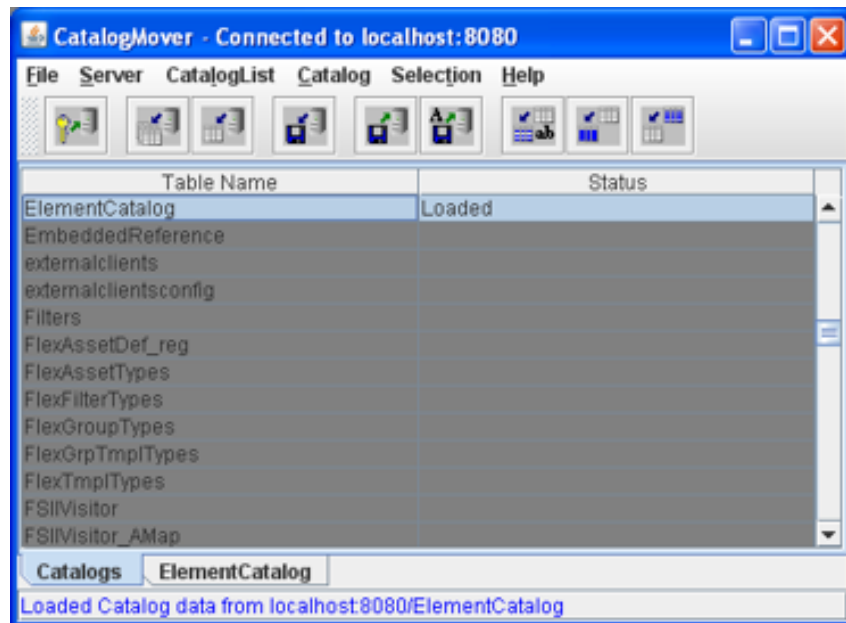
Select the **ElementCatalog** tab to view all rows in the table, and to select specific rows for export.

- **Refresh**: Update the loaded tables from the WebCenter Sites database.

- **Auto Import Catalog(s)**: Import a previously exported ZIP file.

- **Import Catalog**: Import into the local database a table that was exported from another WebCenter Sites database.

- **Export Catalog Rows**: Export the selected rows in the loaded table.

Selection Menu

- **Select All Rows**: Select all rows in the currently displayed table.

- **Deselect All Rows**: Deselect all rows in the currently displayed table.

- **Select Rows By SubString**: Select rows in the currently displayed table by typing a portion of any field value string that uniquely identifies a set of rows.

Help Menu

- **About**: Display version information about the WebCenter Sites installation.

82.3.5 Exporting Tables

Exporting is the process of retrieving table rows and their content from the database and saving them in local HTML files and associated data directories. CatalogMover creates one HTML file per table.

This section includes the following topics:

- Exporting Selected Table Rows

- Selecting Rows for Export

- Exporting to a ZIP File

82.3.5.1 Exporting Selected Table Rows

To export selected table rows follow these steps:

1. Connect to WebCenter Sites as described in Connecting to WebCenter Sites.

2. Choose **CatalogList** then **Load** to display a list of all tables in the database.

3. Choose **Catalog** then **Load** to load a table, and select rows as described in Selecting Rows for Export.

4. Choose **Catalog** then **Export Catalog Rows**.

   A dialog opens prompting you to specify a directory for the HTML file containing the exported rows.

5. Navigate to your directory of choice, and click **Save**.

   CatalogMover exports the selected rows to your selected directory.

82.3.5.2 Selecting Rows for Export

You can select specific rows for export in a loaded table by clicking them, or you can search for specific rows by substring.

To search for and select rows according to a substring:

1. Choose **Selection** and then **Select Rows By SubString**.

   The Catalog Mover dialog opens.

   Figure 82-5   Catalog Mover Dialog
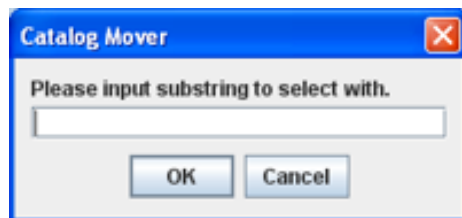
2. In the text field, enter the substring you want to locate. For example, to search the ElementCatalog primary key for all rows with *folder* in the element name, enter `folder` and click **OK**.

   CatalogMover searches the table and selects the rows that match your substring query against the primary key for the table.

Figure 82-6   CatalogMover Results

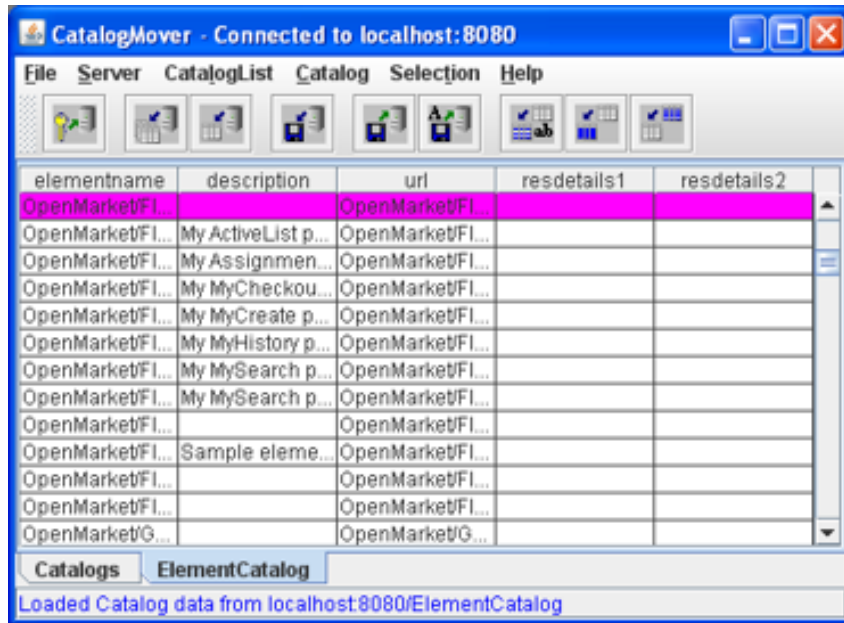**Note:**

Selecting rows by substring only works for the left-most column in the table. However, you can change column positions so that any column can become the left-most column. To do this, simply click and drag the column header.

82.3.5.3 Exporting to a ZIP File

You can select several rows from several tables and export them to a ZIP file on the local computer from which you are running CatalogMover. After you create the ZIP file, import the contents of the file into server tables.

To export a ZIP file with CatalogMover:

1. Choose **CatalogList** then **Load** to display a list of all tables in the database.

2. Choose **Catalog** then **Load** to load a table, and select rows as described in Selecting Rows for Export.

3. Choose **Catalog** then **Export Catalog Rows**.

   The Select Base Directory for Export dialog opens.

Figure 82-7 Select Base Directory for Export Dialog
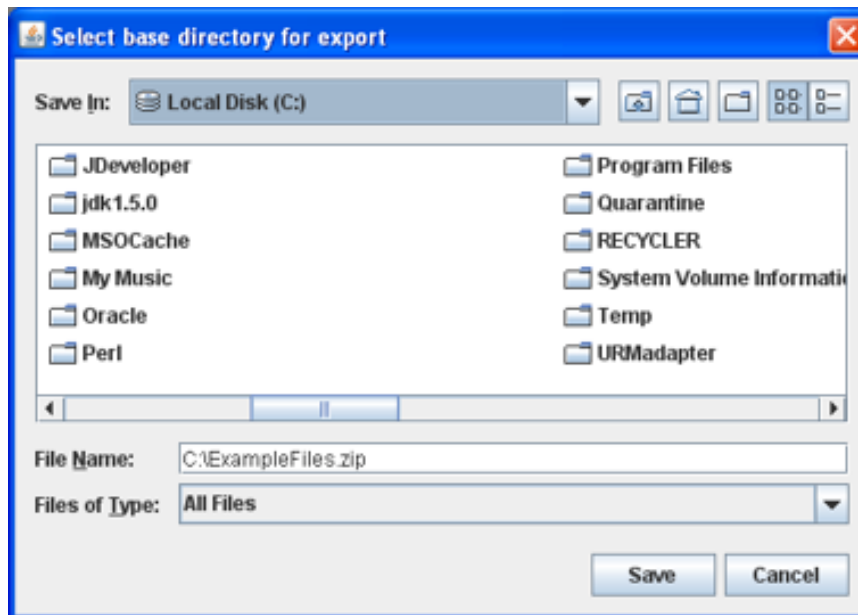
4. Navigate to the directory where you want to save the ZIP file.

5. In the **File Name** field, enter a name for the files and type a ZIP file extension.

6. Click **Save**. The rows you selected from all of the tables are exported to a ZIP file in the directory you chose.

82.3.6 Importing Tables

Importing is the process of sending locally stored HTML files and the associated data to the server. You can select a particular HTML file to import, or you can choose to import all HTML files.

This section includes the following topics:

- Importing HTML Files Previously Exported

- Importing a Previously Exported ZIP File

- Merging Existing CatalogMover Files

- Replacing Existing CatalogMover Files

82.3.6.1 Importing HTML Files Previously Exported

To import HTML files that have been previously exported from another table:

1. Connect to the WebCenter Sites installation you want to import the HTML files to.

2. Choose **CatalogList** then **Load** to display a list of all tables in the database.

3. Choose **Catalog** then **Import Catalog**.

4. Navigate to the HTML file containing the previously exported table rows.

5. Select the HTML file and click **Open**.

The Select Base Directory for Export dialog opens.

6. To import new table rows that do not currently exist, enter the information in the **Catalog Data Directory** and the **Catalog ACL List** fields.

   To replace existing table rows with the imported table rows, leave these fields blank.

7. Click **OK**. The table rows contained in the previously export HTML file are imported into the WebCenter Sites database to which you are connected.

   A dialog opens, listing the table rows that were imported.

   Note:

   The new tables are automatically created when you import tables that do not exist on the server to which you are connected.

82.3.6.2 Importing a Previously Exported ZIP File

You can import table rows stored in an exported ZIP file to your server using CatalogMover.

To import a previously exported ZIP file:

1. While connected to your database, choose **Catalog** then **Auto Import Catalogs**.

2. In the dialog, navigate to the directory where you previously exported the table rows. To see the ZIP file, change the **Files by Type** menu to **all files**.

3. Select the ZIP file and click **Save**. The rows contained in the ZIP file are automatically imported to your database.

82.3.6.3 Merging Existing CatalogMover Files

To merge CatalogMover files follow these steps:

1. Connect to the WebCenter Sites installation you want to import the HTML files to.

2. Choose **CatalogList** then **Load** to display a list of all tables in the database.

3. Choose **Catalog** then **Load** to load a table, and select the rows that you want to merge into another file, as described in Selecting Rows for Export.

4. Choose **Catalog** then **Export Catalog Rows.**

5. Navigate to the HTML file you want to merge the rows with. Click **Save**.

   The Overwrite dialog opens.

6. Click **Update existing exported data**. CatalogMover merges the exported rows into the HTML file you selected.

82.3.6.4 Replacing Existing CatalogMover Files

To replace CatalogMover files follow these steps:

1. Connect to the WebCenter Sites installation you want to import the HTML files to.

2. Choose **CatalogList** then **Load** to display a list of all tables in the database.

3. Choose **Catalog** then **load** to load a table, and select the rows that you want to merge into another file, as described in Selecting Rows for Export.

4. Choose **Catalog** then **Export Catalog Rows**.

5. Navigate to the HTML file you want to merge the rows with and click **Save**.

   The Overwrite dialog opens.

6. Click **Replace existing exported data**.

   CatalogMover replaces rows in the HTML file you selected with the exported rows.

82.3.7 Command Line Interface

Parameters described in the table below allow CatalogMover to perform functions without displaying a GUI. The parameter is followed by a space followed by the value.

Table 82-1   Command Line Interface Parameters

Parameters	Description
-h	Display command line parameters
-u username	User name
-p password	Password
-s servername	Servername to connect
-b baseurl	Base URL: either `http://($host)/cgi-bin/gx.cgi/AppLogic +FTCatalogManager (NAS)` or `http://($host)/servlet/ CatalogManager(WebLogic)`
-t table	Table name: Used when exporting to designate tables to export, use multiple -t parameters to export multiple tables.
-x function	Function to perform: Legal values are `import`, `import_all`, `export`, `export_all`.
-d directory	Directory: When exporting, directory to contain exported tables. When importing all, directory containing all tables to import.
-f filename	File containing table to import: Can either be an HTML file or a ZIP file generated by export.
-c directory	Upload directory to be used if creating a table.
-a aclone,acltwo,...	ACL list: Comma-separated list of ACLs to be used if creating a table.

82.4 Property Management Tool

The Property Management Tool includes an easy-to-use form that you use when you want to search for, view, modify, and add properties in the `wcs_properties.json` file.
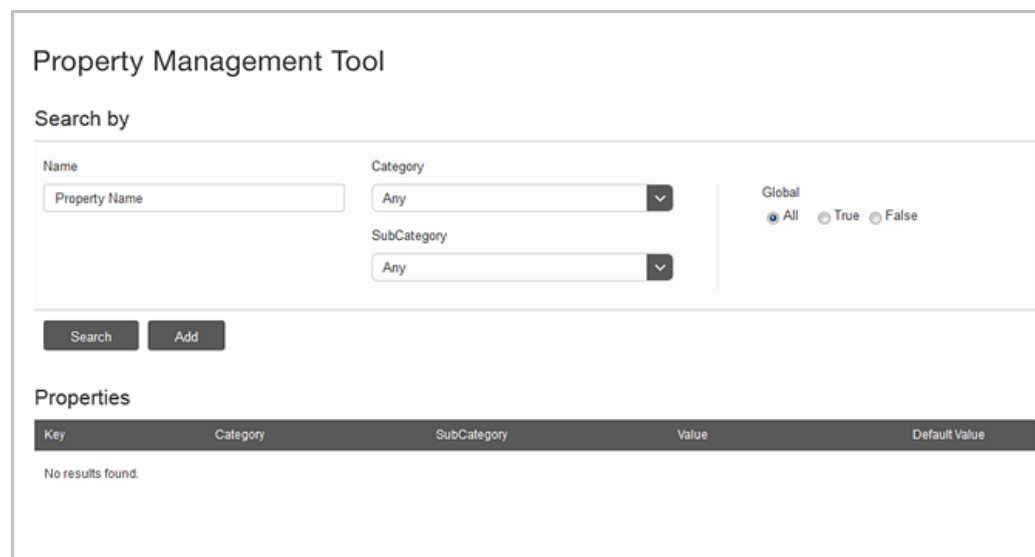
Topics:

- Accessing the Property Management Tool

- Setting Properties

- Adding Properties to the wcs_properties.json File

82.4.1 Accessing the Property Management Tool

1. Log in to WebCenter Sites, select the name of the site, and then select the **Admin** interface icon.

2. In the **General Admin** tree, expand the **System Tools** node.

3. Under the **System Tools** node, double click **Property Management**.

The Property Management Tool opens.

Figure 82-8   Property Management Tool

The Search by section of the Property Management Tool enables you to search for the full or partial name of a property. You can also search by the Category assigned to the property in the `wcs_properties.json` file.

When you click **Search**, the Properties section of the form lists all the properties that match the search criteria you enters in the Search by section of the form.

The **Key** column displays the name of the property, the **Category** column displays the category assigned to the property, the **SubCategory** column displays the sub category (if any) assigned to the property, the **Value** column shows the value to which the property is currently set. and the **Default Value** column displays the recommended value for the property.

> **Note:**
>
> The `wcs_properties.json` file contains a release number string, `ft.version`, which contains a value such as 4.0.0. that is set by WebCenter Sites.
>
> Do not modify this property, it is for reference only.

82.4.2 Setting Properties

To set WebCenter Sites properties on your system:

1. If necessary, open the Property Management Tool.

2. In the Search by section, do one of the following:

   - In the **Name** field, enter the name of the property you want to modify.

   - In the **Category** drop-down menu, select the Category assigned to the property in the `wcs_properties.json` file.

3. Click **Search**.

4. In the **Key** column, select the name of the property.

5. In the **Value** field, enter the new value.

6. Click **Save**.

7. Repeat steps 2 through 6 for all the properties you want to change.

8. Stop and restart the application server to apply the changes.

82.4.3 Adding Properties to the wcs_properties.json File

In certain cases, you might need to add properties to the `wcs_properties.json` file. In such cases, use the Property Management Tool to add properties to this file.

1. Open the Property Management Tool.

2. In the Search by section of the form, click **Add**.

3. Enter values for the new property. See Adding Properties in the *Property Files Reference for Oracle WebCenter Sites*.

4. Click **Save**.

5. Stop and restart the application server so the new property can take effect.

82.5 About Importing with XMLPost

You use the XMLPost utility to import data into the WebCenter Sites database. This utility is based on the WebCenter Sites `FormPoster` Java class and it is delivered with the WebCenter Sites base product. It imports data using the HTTP POST protocol.

To import assets, you use XMLPost with posting elements that are delivered with WebCenter Sites.

Topics:

- Importing Assets of Any Type

- Importing Flex Assets

83

Understanding White Space and Compression

When WebCenter Sites streams a text page, the page may contain a significant amount of white space (spaces, carriage returns, and tabs) that have no effect on the data that is consumed by the client. The white space is visible when the source code is viewed by the consumer. Furthermore, excessive white space needlessly increases the size of the response, which ultimately increases bandwidth use. So, it's a good practice to eliminate the white space whenever possible.

Topics:

- White Space and JSP

- White Space and XML

- Compression

- JSP Design

83.1 White Space and JSP

The JSP specification needs you to preserve the white space.

Thus, a page that looks like this:

```
<%@ page import="my class name"%>
<%@ page import="my class 2"%>
<cs:ftcs>
<p>Hello world!</p>
</cs:ftcs>
```

has three carriage returns and a tab preceding the `<p>` because the text is displayed on third line after the JSP has been interpreted. With more complicated pages, the problem is compounded.

83.2 White Space and XML

WebCenter Sites XML processing language, being a proprietary set of XML-compliant tags, does not adhere to the white space preserving rules of JSP.

As such, a WebCenter Sites XML page like this:

```
<? XML version 1.0 ?>
<FTCS>
<p>Hello World!</p>
</FTCS>
```

displays `<p>` as the first characters of output, because our XML parser strips all of the white space. (An exception is if XML debug is enabled, in which case all white space is preserved).

83.3 Compression

White space is an artifact of writing well-formatted code. Its presence is a side effect of programming practices that benefit the developer. The impact on the consumer and the customer is minimal except for bandwidth. To address bandwidth, you can compress the output of all text-based pages. You need to be on the server-side to compress the output. The consumer's user-agent (browser) performs decompression.

The compression/decompression is completely transparent to the end user. This sort of compression can yield up to an 80% reduction in bandwidth use.

One commonly-used compression mechanism is the mod-gzip extension to the Apache web server. This module automatically gzips all output to the user agent, provided that it can decompress it. Configuration is minimal and its effectiveness is quite high. It can be obtained from SourceForge (`http://sourceforge.net/projects/mod-gzip/`). Similar tools are available for other common web servers, such as IIS.

Another possibility is to do the compression at the application server layer, and leave the web server alone. This is best done by connecting a standard servlet filter to Satellite Server (or to WebCenter Sites if Satellite Server is not being used). The servlet filter is invoked in a prescribed order before or after the invocation of the specified servlet (or both), and during invocation it can compress the output before sending it to compatible user-agents, exactly the same way mod-gzip works. One such compression filter can be found at SourceForge (`http://sourceforge.net/projects/pjl-comp-filter/`).

If you need assistance with compression, contact Oracle Consulting Services.

83.4 JSP Design

If compression is not an option, consider altering your JSP pages in a way that white space doesn't occur.

You can do this by changing the code above to the following:

```
<%@ page import="my class name"
%><%@ page import="my class 2"
%><cs:ftcs><p>Hello world!</p></cs:ftcs>
```

While this is not as elegant (or readable), it results in page output without any white space whatsoever before the `<p>` tag. An intermediate solution may be something like the following:

```
<%@ page import="my class name"
%><%@ page import="my class 2"
%><cs:ftcs>
<p>Hello world!</p>
</cs:ftcs>
```

For extensive examples of how to address white space issues in JSP, refer to our WebServices elements in the ElementCatalog. They are included with WebCenter Sites.

84

Using WebCenter Sites URL Assemblers

WebCenter Sites URL assemblers manage URL assembly and disassembly. You can use the interface that they provide to define the appearance of URLs.

Topics:

- About WebCenter Sites URL Assemblers

- Assemblers Installed with WebCenter Sites

- Working with Assemblers

- Vanity URL Links in a Web Page

84.1 About WebCenter Sites URL Assemblers

You can use URL assemblers along with URL generation tags to generate WebCenter Sites URLs and to disassemble the URLs.

URL Assemblers are the legacy method of URL management. Vanity URLs are the current method. See Vanity URL Links in a Web Page.

Topics:

- URL Assembly

- Assembler Discovery and Disassembly

- URL Assembly and Disassembly Using GET and POST Requests

84.1.1 URL Assembly

WebCenter Sites URL generation tags (`<satellite.link>`, `<satellite.blob>`, `<render.getpageurl>`, `<render.getbloburl>`, `<render.satelliteblob>`, `<rendergettemplateurl>`) are used to construct a link to a WebCenter Sites resource, such as a page or a blob. The data, such as tag attributes and nested argument tags which you specify when using the tag, is converted into an abstract object called a **URL definition**. The URL definition is passed into the **URL assembler**. The URL assembler then converts the definition into a string URL that is returned.

Two assemblers are installed with WebCenter Sites, but you have the option of creating your own assemblers to directly control the appearance of your URLs. Before you can use the assemblers you create, you must first register them with WebCenter Sites.

The assembler that is configured as the default is used to create all WebCenter Sites URLs. You can change the default assembler. You can also override the use of this default assembler in individual link tags.

84.1.2 Assembler Discovery and Disassembly

Because an assembler can create a URL in any form that the assembler's author dictates, it may be impossible for the URL to be decoded into parameters by an application server when an assembled link is requested. For decoding to take place, the assembler must be able to disassemble the string URL into its definition. Assemblers are therefore reversible, that is, capable of disassembling any URLs that they assembled.

URL created using an assembler other than the default one can't be disassembled by the default assembler. At that point, the next highest ranked assembler attempts to disassemble the URL. If it succeeds in creating a definition, then the assembler engine has discovered its assembler, and the definition is converted into parameters for processing. If the next highest ranked assembler fails to disassemble the URL, the third highest ranked assembler is called upon to disassemble it. This process continues until the URL is successfully disassembled. Note that this process requires an assembler to be able to recognize the URLs it assembled as its own, and all other URLs as foreign.

See Creating Assemblers and Registering and Ranking Assemblers.

84.1.3 URL Assembly and Disassembly Using GET and POST Requests

URL assemblers are only invoked on GET requests. They are not invoked on POST requests. For example, when accessing a page with a GET request, the URL assembler is invoked to disassemble the URL. It then provides the appropriate parameters that WebCenter Sites requires to open that page (such as c, cid, and pagename) by adding them to the definition (if they do not exist in the definition). However, when a request is POSTed, such as a form with method=post, the URL assembler is not invoked to disassemble the URL. Therefore, the parameters WebCenter Sites requires to open the page must be part of the post request itself.

This can be accomplished by encoding the following tag into the page's template or element (replacing the sample values with the parameters WebCenter Sites requires for the page's URL):

```
<satellite:form method="post" id="assetid">
<render:gettemplateurlparameters list="args" ... /><!-- add all parameters
that are normally part of a URL -->
        <ics:listloop listname="args">
                <input type="hidden" name="<string:stream list="args"
column="name"/>" value="<string:stream list="args" column="value"/>"/>
        </ics:listloop>
```

84.2 Assemblers Installed with WebCenter Sites

The two assemblers that are installed with WebCenter Sites are Query Assembler and QueryAsPathInfo Assembler. While the Query Assembler uses query strings to create URLs, the QueryAsPathInfo Assembler encodes the query strings and appends them to the end of servlet names.

Topics:

- Query Assembler

- QueryAsPathInfo Assembler

84.2.1 Query Assembler

The Query Assembler creates URLs with query strings. It is the default assembler, and it is automatically registered in WebCenter Sites. Therefore, until you make any modifications (such as changing the default assembler or overriding the default in link tags), Query Assembler is used to generate all URLs.

84.2.2 QueryAsPathInfo Assembler

The QueryAsPathInfo Assembler does not use query strings. Instead, the QueryAsPathInfo Assembler encodes the query string and appends it to the end of the servlet name. The benefit of this assembler is that it creates URLs that can be indexed by search engines. The QueryAsPathInfo Assembler is not automatically registered with WebCenter Sites.

84.3 Working with Assemblers

You can create and register your own assemblers and modify link tags to override the use of the default assembler.

Topics:

- Creating Assemblers

- Registering and Ranking Assemblers

- Link Tags Modification

84.3.1 Creating Assemblers

The WebCenter Sites URL Assembly module enables you to create your own assemblers. This option gives you direct control of the appearance of your URLs.

To create an assembler:

1. Write a Java class that implements the `com.fatwire.cs.core.uri.Assembler` interface. For information about this class, see the *Oracle Fusion Middleware Java API Reference for Oracle WebCenter Sites*.

2. Compile the class into a `.jar` file.

3. Deploy your class into the WebCenter Sites web application and the web application for each remote Satellite Server you have installed.

   This usually means copying the `.jar` file you just created into your web application's `WEB-INF/lib` folder. For remote Satellite Servers, this means copying it to the `resin/webapp/ROOT/WEB-INF/lib` folder.

4. Register your new assembler in the `wcs_properties.json` file on WebCenter Sites and all of your remote Satellite Servers. (See Registering and Ranking Assemblers.)

5. Restart WebCenter Sites and all of your remote Satellite Servers.

84.3.2 Registering and Ranking Assemblers

Before an assembler can be used to create URLs, it must first be registered with WebCenter Sites. The registration is done by listing assembler class names with

corresponding short forms in a property file. The registration also includes a ranking that indicates in which order the assemblers should be used.

To register an assembler:

1. In the Admin interface, open the Property Management Tool.

2. In the **Category** drop-down menu, select **ServletRequest** to access the assembler properties.

3. Click **Search**.

4. Specify the `classname` and `shortform` of the assembler you want to register.

   The third element in the property name indicates the ranking of the assembler. The assembler with the ranking of `0` is the highest ranked (and default) assembler, the assembler with the ranking of `1` is the next highest ranked, and so on.

   To configure the new assembler to be the default assembler, enter the `classname` and `shortform` values in the properties that have 1 as their ranking.

   For example, the syntax to register the QueryAsPathInfo assembler as the default assembler would be as follows:

   Table 84-1   Assembler Properties

Property Name	Property Value
`uri.assembler.0.classname`	`com.fatwire.cs.core.uri.QueryAsPathInfoAssembler`
`uri.assembler.0.shortform`	`pathinfo`
`uri.assembler.1.classname`	`com.fatwire.cs.core.uri.QueryAssembler`
`uri.assembler.1.shortform`	`query`

   > **Note:**
   >
   > Ensure the Query Assembler is always registered, even if you have lowered its ranking. The Query Assembler must be registered with the `shortform` value of query.

5. Depending on the ranking of the new assembler, you may have to adjust the rankings of the other assemblers. Verify that all of the assemblers are configured and ranked correctly in the property file. If they are not, make any necessary changes.

6. After you change the value of an assembler, click **Save** before modifying the value of a different assembler.

7. Repeat steps 5 through 6 for each remote Satellite Server you have installed.

8. Restart WebCenter Sites and all of your remote Satellite Servers.

84.3.3 Link Tags Modification

WebCenter Sites link tags can be modified to use an assembler other than the default assembler. The link tags accept an attribute, `assembler`, which take an assembler short form as a value. For example, to override the default assembler with the QueryAsPathInfo assembler in an individual link tag, the syntax would be as follows:

```
<satellite:link pagename=example assembler=pathinfo />
```

84.4 Vanity URL Links in a Web Page

If an asset has a vanity URL for the template that is passed in the name argument, then the `<render:gettemplateurl>` tag returns the vanity URL. If the asset does not have a vanity URL, then the tag returns the WebCenter Sites URL.

The `<render:gettemplateurl>` tag can be used to generate a vanity URL link. Thus any redirected URL always returns the vanity URL if present. That way, if an existing link is accessed (for example, from a bookmark), then all subsequent URLs are vanity URLs.

To generate a vanity URL for a specific WebRoot, an additional optional parameter called `hostparam` should be passed to `render: gettemplateurl`. The value of the `hostparam` should match the `hostname` attribute of WebRoot. This can be used to generate a link to another website or subdomain within the same website. For example:

```
<render:gettemplateurl outstr="pageURL" webrootname=<name of WebRoot used for
retrieving URL> tname='<%=ics.GetVar("template")%>' args="c,cid" />
```