Oracle® WebCenter Sites

Developer's Guide for the Community Application

11*g* Release 1 (11.1.1)

February 2012

ORACLE®

Oracle WebCenter Sites Developer's Guide for the Community Application, 11*g* Release 1 (11.1.1)

Contents

4 Securing the Community Application

5 Translating the Community Application's Functionality into Different Languages

6 Monitoring Community Application Performance

A   Guidelines for Maintaining the Community Application

Index

Preface

This guide begins with an overview of the Community application and its developers. It continues to the process of integrating the Community application with Facebook, Twitter, and Janrain. This guide also provides information about customizing and securing the Community application, as well as translating its functionality into various languages. Finally, this guide includes guidelines and tips for maintaining the Community application experience.

Applications discussed in this guide are former FatWire products. Naming conventions are the following:

- *Oracle WebCenter Sites* is the current name of the product previously known as *FatWire Content Server*. In this guide, *Oracle WebCenter Sites* is also called *WebCenter Sites*.

- *Oracle WebCenter Sites: Community* is the current name of the application previously known as *FatWire Community Server*. In this guide, *Oracle WebCenter Sites: Community* is also called the *Community* application.

- *Oracle WebCenter Sites: Web Experience Management Framework* is the current name of the environment previously known as *FatWire Web Experience Management Framework*. In this guide, *Oracle WebCenter Sites: Web Experience Management Framework* is also called *WEM Framework*.

- *Oracle WebCenter Sites: Satellite Server* is the current name of the application previously known as *FatWire Satellite Server*. In this guide, *Oracle WebCenter Sites: Satellite Server* is also called *Satellite Server*.

The Community application integrates with Oracle WebCenter Sites according to specifications in the *Oracle WebCenter Sites 11g Release 1 (11.1.1.x) Certification Matrix*. For additional information, see the release notes for the Community application. Check the WebCenter Sites documentation site regularly for updates to the *Certification Matrix* and release notes.

Audience

This guide is for Community application developers who are familiar with the default functionality of the Community application.

Related Documents

For more information, see the following documents:

- *Oracle WebCenter Sites Installation Guide for the Community Application*
- *Oracle WebCenter Sites User's Guide for the Community Application*

- *Oracle WebCenter Sites Administrator's Guide*

Conventions

The following text conventions are used in this document:

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

Third-Party Libraries

Oracle WebCenter Sites and its applications include third-party libraries. For additional information, see *Oracle WebCenter Sites 11gR1: Third-Party Licenses*.

1

Introduction to Oracle WebCenter Sites: Community

Oracle WebCenter Sites: Community (the Community application) is a social computing application designed to gather visitors' comments, reviews, and ratings on web site content. The Community application also enables administrators and moderators to create and manage polls that can be used to survey site visitors about desired topics. This chapter includes the following sections:

- Section 1.1, "Technical Overview of Oracle WebCenter Sites: Community"
- Section 1.2, "Prerequisites"

1.1 Technical Overview of Oracle WebCenter Sites: Community

The Community application's functionality such as comments, reviews, and polls is developed as separate modules called *widgets*. Each widget can be deployed on a web site by using its JavaScript code snippet, which renders the widget in a browser. The ability to deploy widgets in a browser makes deployment simple and platform-independent. The widget deployment approach also facilitates high scale maintenance and upgrades.

Community widgets are compatible with most server-side and backend technologies, and therefore, there are no major deployment prerequisites that developers must know of. Since the Community application is an add-on for WebCenter Sites, it can be developed separately from WebCenter Sites.

The use of JavaScript enables developers to create easy-to-use and feature-rich widget interfaces. Developers can build dynamic screens that are responsive to user actions and do not require the whole web page to be re-loaded. As soon as a widget is deployed on a web page, it starts rendering and loading dynamic content such as lists of comments, reviews, or particular poll information from the data repository. The data for a widget is loaded using JavaScript, with the help of a remote scripting technology similar to AJAX. This technology also facilitates the deployment of the Community application on a domain other than the original web site domain. Figure 1–1 shows the post-deployment web site environment.

Figure 1–1   Client Web Site Environment After Deploying Community Functionality
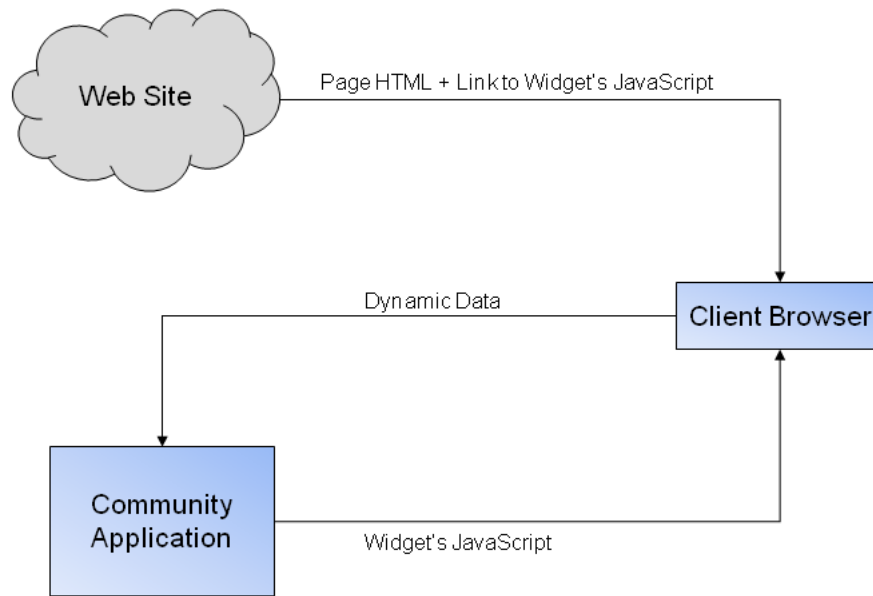
To become familiar with the Community object data model on which the Community widgets are based, see Section 3.1, "Overview of Community Data Model."

1.2 Prerequisites

Developers must understand how the Community application is installed and know the Community interface. To effectively use this guide, developers must have experience with the following:

- WEM Framework, REST API, and WEM Admin interface.

- Oracle WebCenter Sites asset model and Oracle WebCenter Sites: Satellite Server for caching.

- Central Authentication Service (CAS) used for Web Single Sign-On.

- Deployment of web applications on Oracle WebLogic Server, WebSphere Application Server or Tomcat application server.

2

Integrating the Community Application With Social Networking Services

The Community application's widgets are designed to leverage the authentication APIs of Facebook, Twitter, and Janrain on your web site. Community widgets enable the visitors of your web site (or blogs) to use their existing social networking profiles to provide feedback on your company's content and to share the same content on their social networking profiles. This chapter describes how to integrate the Community application with Facebook, Twitter, and Janrain.

This chapter includes the following sections:

- Section 2.1, "About Authentication Plug-Ins"
- Section 2.2, "Integrating with Facebook"
- Section 2.3, "Integrating with Twitter"
- Section 2.4, "Integrating with Janrain"
- Section 2.5, "Enabling Social Networking Services on WebSphere Application Server"

2.1 About Authentication Plug-Ins

Developers can leverage third-party authentication providers in the following ways:

- Use the built-in support of Facebook and Twitter, which is available out-of-the-box with the Community application.
- Use the authentication hub, Janrain, to enable access to most prevalent providers available online. Janrain is a SaaS solution that provides integration services with a number of online authentication providers such as Facebook, Twitter, Google, and so on. The more services supported, the more chances that the site visitor has an identity in one of these services, which they may use to log in.

2.2 Integrating with Facebook

On a web site, visitors' Facebook identities are authenticated via a trusted bridge, which is established between the Community application and Facebook at each login attempt. To enable this bridge, you must first create a Facebook application using the Community application's production site URL. Then, configure the Facebook authentication properties on the application server on the production system of the Community application. This section describes how to:

- Create a Facebook Application for the Community Application

- Configure Facebook Application's Authentication Settings on the Community Application

2.2.1 Create a Facebook Application for the Community Application

> **Note:** Steps and screens shown in this section may not match the Facebook interface at the time when you create an application. If you see new fields and require help, contact product support.

To create a Facebook application for the Community application:

1. Log in to Facebook to access the "Apps" page: https://developers.facebook.com/apps

2. On the top right corner of the page, click **Create New App** to display the "New App" dialog box.

3. In the "App Display Name" field, enter a name for the Community application.

4. Select the **I agree to the Facebook Platform Policies** check box, then click **Continue**.

Figure 2–1   New App - Facebook

5. On the "Security Check Required" page, in the "Text in the box" field, enter the displayed CAPTCHA text. Then, click **Submit**.

6. Under "Basic", copy the values of "App ID" and "App Secret" parameters (Figure 2–2). You will need these values while configuring the Facebook authentication properties on the application server.

Figure 2–2   App ID And App Secret

7. Under "Select how your app integrates with Facebook", click the gray text next to "Website" to display the "Site URL" input field, as shown in Figure 2–3.

Figure 2–3   Select How Your App Integrates With Facebook

8. In the "Site URL" field, enter the Community application's production address, then click **Save Changes**.

2.2.2 Configure Facebook Application's Authentication Settings on the Community Application

To configure Facebook application's authentication properties on the Community application server:

1. In the application server home on the production system, navigate to the `cos.war/WEB-INF/classes` directory. For example, if you are using Tomcat, go to `<App__Server_Home>/webapps/cos/WEB-INF/classes`.

2. Open the `setup_auth.properties` file, then scroll down to the "Facebook Application Settings" section.

3. For `widgets.external_auth.facebook.attrs.client_id` and `widgets.external_auth.facebook.attrs.client_secret` configuration properties, enter the application ID and application secret values you copied at the time of creating the Facebook application.

    The "Facebook Application Settings" section should look like this:

    ```
    ## Facebook Application Settings
    ######################################
    #
    # Facebook application id
    #
    widgets.external_auth.facebook.attrs.client_id=11111111111111111111
    #
    # Facebook application secret
    #
    widgets.external_auth.facebook.attrs.client_secret=571bf600e8d23
    ```

4. After saving the configuration file, restart the application server.

5. Open the Community application on the management side as a general administrator: `http://host:port/cs/login`.

6. Choose the desired site.

7. From the "Login Bar" menu, choose **Configure**.

8. In the "Login and Social Settings" section, under "Enabled native social integration", select the **Enable Facebook login** check box (Figure 2–4), then click **Save**.

Figure 2–4   Enable Facebook Login

9. Restart the management and production environments.

10. Access the web site on which the widget is deployed.

    The Facebook login button is displayed (Figure 2–5) in the Community widget's "Login" screen.

Figure 2–5   Facebook Login Button

> **Note:** If your Community application is running on Oracle WebLogic Server or Tomcat application server, then the procedures described in Section 2.2.1, "Create a Facebook Application for the Community Application" and Section 2.2.2, "Configure Facebook Application's Authentication Settings on the Community Application" are all you require to integrate the Community application with Facebook. However, on a WebSphere Application Server instance, you must also perform the export/import procedures described in Section 2.5, "Enabling Social Networking Services on WebSphere Application Server" to enable communication between the Community application and this social networking service.

2.3 Integrating with Twitter

You can integrate the Community application with Twitter just like you integrated it with Facebook. That is, first create a Twitter application using the production site URL, then update the authentication properties on the application server to include the Twitter application's required information.
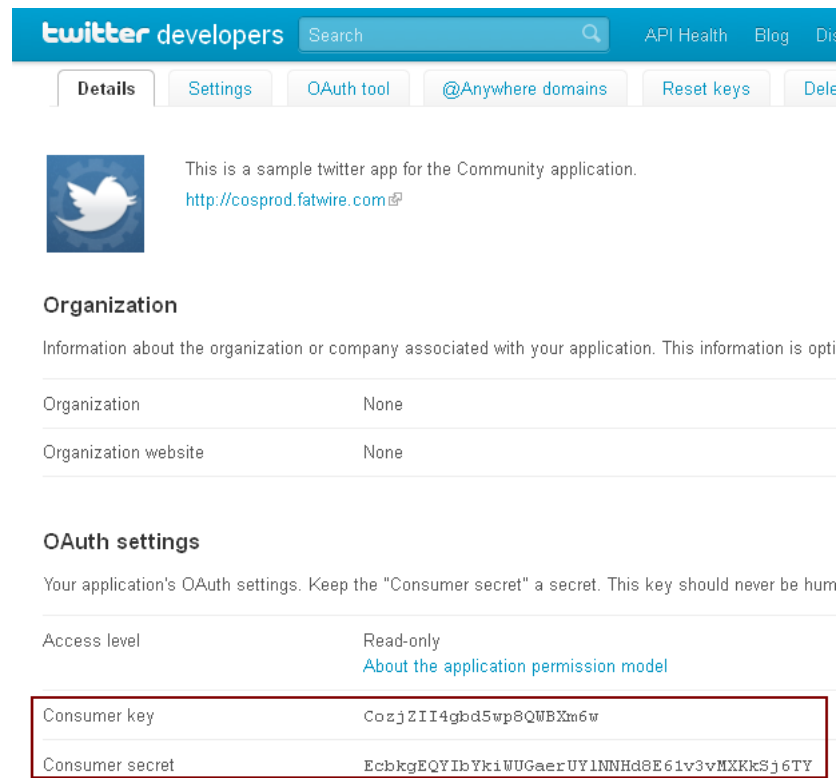
This section describes how to:

- Section 2.3.1, "Create a Twitter Application for the Community Application"
- Section 2.3.2, "Configure Twitter Application's Authentication Settings on the Community Application"

2.3.1 Create a Twitter Application for the Community Application

To create a Twitter application:

1. Log in to Twitter's developers central at `https://dev.twitter.com/`.

2. Under "Create applications that integrate Twitter", click **Create an app**.

3. Under "Application Details", enter the required information as follows:

   a. In the "Name" and "Description" fields, enter a name for the Community application and a description.

   b. In the "WebSite" field, enter the Community application's production server URL. For example, `http://production.acme.com`.

   c. In the "Callback URL" field, enter the production host location of the Community application, which is the same as the URL in step b.

   d. Select the **Yes, I agree** check box.

   e. Under "CAPTCHA", enter the words you see on the screen.

   f. Click **Create your Twitter application**.

4. From the "Details" page, copy the values of "Consumer key" and "Consumer secret" properties displayed in the "OAuth settings" section (Figure 2–6). You will need these values to configure the authentication properties on the application server.

Figure 2–6 OAuth Settings

2.3.2 Configure Twitter Application's Authentication Settings on the Community Application

To configure the authentication properties on the Community application's server:

1. In the application server home on the production system, navigate to the `cos.war/WEB-INF/classes` directory. For example, if you are using Tomcat, go to `<App_Server_Home>\webapps\cos\WEB-INF\classes`.

2. Open the `setup_auth.properties` file, then scroll down to the "Twitter Application Settings" section.

3. For `widgets.external_auth.twitter.attrs.consumer_key` and `widgets.external_auth.twitter.attrs.consumer_secret` configuration properties, enter the consumer key and consumer secret values you copied earlier.

    The "Twitter Application Settings" section should look like this:

    ```
    ## Twitter Application Settings
    # Twitter consumer key
    #
    widgets.external_auth.twitter.attrs.consumer_key=uh5nvtwCg5WjBD9UHM29eQ
    #
    # Twitter consumer secret
    #
    widgets.external_auth.twitter.attrs.consumer_secret=CaYaLDk9IZBConpQVkxk
    ```

4. After saving the configuration file, restart the application server.

5. Open the Community application on the management side as a general administrator: `http://host:port/cs/login`.

6. Choose the desired site.

7. From the "Login Bar" menu, choose **Configure**.

8. In the "Login and Social Settings" section, under "Enabled native social integration", select the **Enable Twitter Login** check box, then click **Save**.

9. Restart the management and production environments.

10. Access the web site on which the widget is deployed.

   The Twitter login button is displayed (Figure 2–7) in the Community widget's "Login" screen on the web site on which the widget is deployed.

Figure 2–7   Twitter Login Button
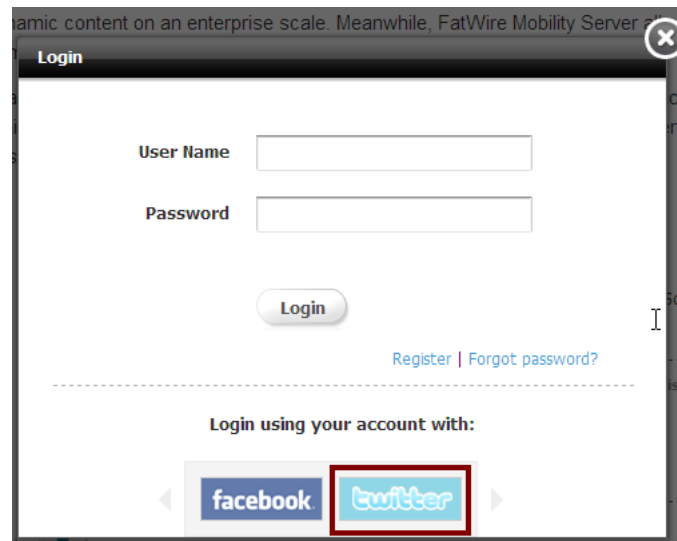
> **Note:**   If your Community application is running on Oracle WebLogic Server or Tomcat application server, then the procedures described in Section 2.3.1, "Create a Twitter Application for the Community Application" and Section 2.3.2, "Configure Twitter Application's Authentication Settings on the Community Application" are all you require to integrate the Community application with Twitter. However, on a WebSphere Application Server instance, you must also perform the export/import procedures described in Section 2.5, "Enabling Social Networking Services on WebSphere Application Server" to enable communication between the Community application and this social networking service.

2.4 Integrating with Janrain

The Community application integrates with Janrain, and Janrain integrates with a number of third-party services. From this list of services, visitors can choose a service that they already use to log in to your site.

To configure Janrain support:

1. Create an account on Janrain's web site.

2. Create a Janrain Application for the Community Application.

3. Configure Janrain Application's Authentication Settings on the Community Application.

2.4.1 Create a Janrain Application for the Community Application

To perform the steps described in this section, you must have a Janrain account. Register your company on the Janrain web site by choosing the "Enterprise" subscription plan.

To create a Janrain application:

1. Go to `www.rpxnow.com`, then from the drop-down list on the toolbar, as shown in Figure 2–8, select **Create Application**.

Figure 2–8   Create Janrain Application
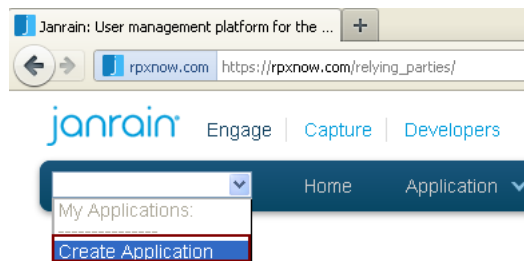
2. Fill in the required fields, then click **Create Application**.

Figure 2–9   Purchase a Plus Application

3. From the "Deployment" menu, choose **Application Settings**.

Figure 2–10   Application Settings
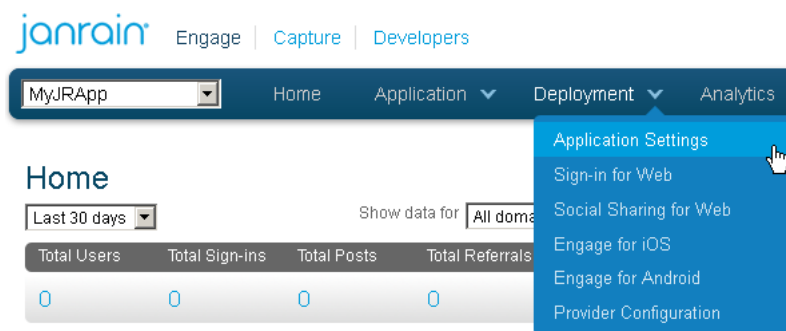
4. In the "Application Info" section, unique values for your newly created application are displayed under "Application Domain", "App ID", and "API Key (Secret)". Copy these values as you will need them when you configure Janrain support on the Community application's production server.

Figure 2–11   Required Application Parameters
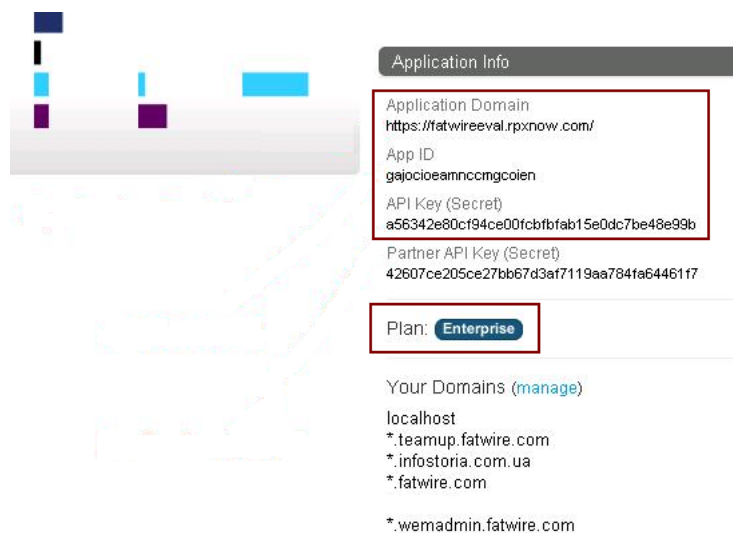
5. In the "Domain Whitelist" box, specify the web site domains on which the Community application widgets will be deployed (Figure 2–12).

Figure 2–12   Domain Whitelist

6. In this step, you will enable the identity providers to be listed on the web site on which Community widgets will be deployed. Therefore, visitors will be able to use one of their existing online identities to add comments and ratings on your web site content.

a. From the "Deployment" menu, choose **Sign-in for Web**.

Figure 2–13   Sign-in for Web
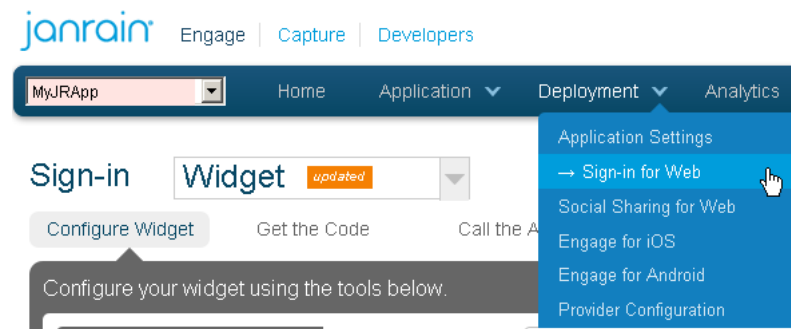
b. Click the **Choose Providers** link, and drag each provider to "Your Widget".

Figure 2–14   Configuration of Identity Providers
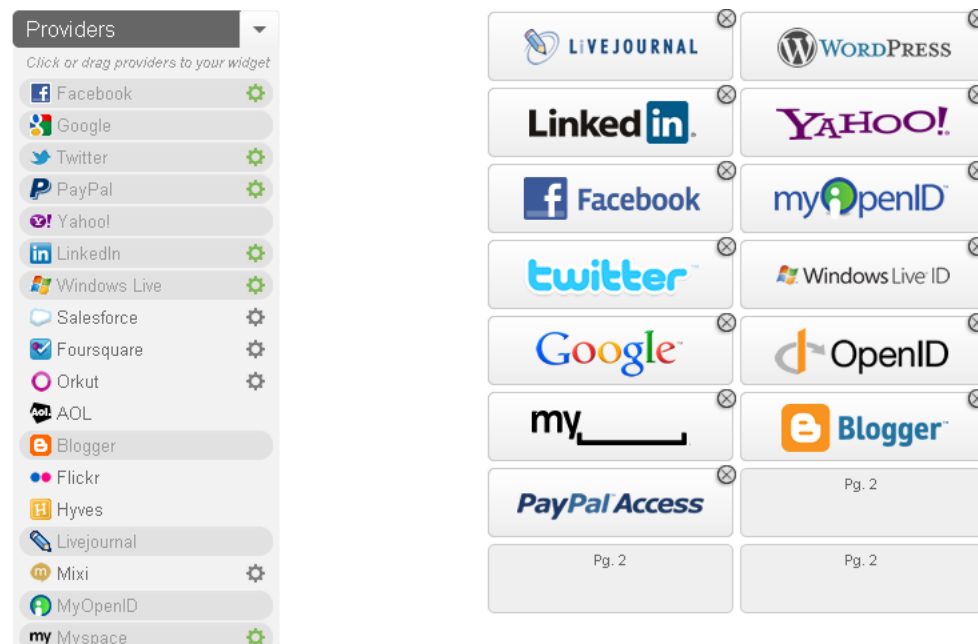
7. Configuring identity providers will allow site visitors to choose a provider for which they already have login credentials. When an identity provider is enabled, its configuration wizard is displayed. For example, Figure 2–15 shows the configuration wizard for Google. Complete the configuration steps for each enabled provider.

Figure 2–15   Google Configuration Via Janrain
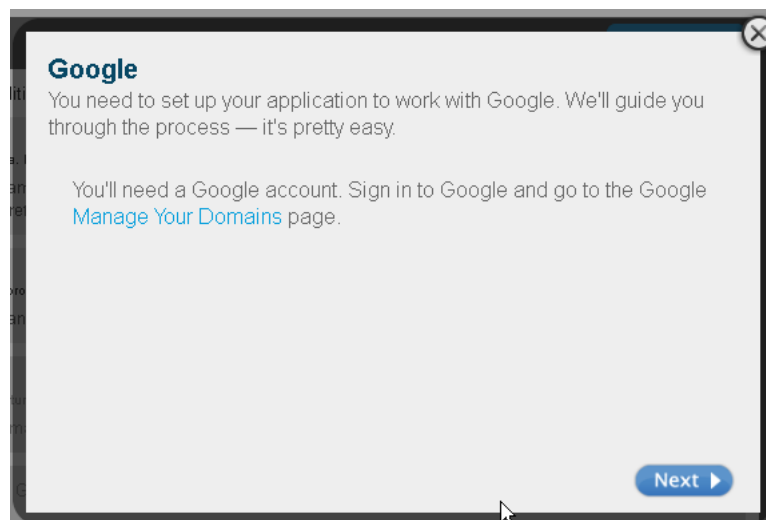
8. Configure social sharing with Janrain. This step will enable your web site visitors to share selected content via their profiles on social networking sites. From the "Deployment" menu, choose **Social Sharing for Web**.

Figure 2–16   Configuration of Social Networking Services Providers
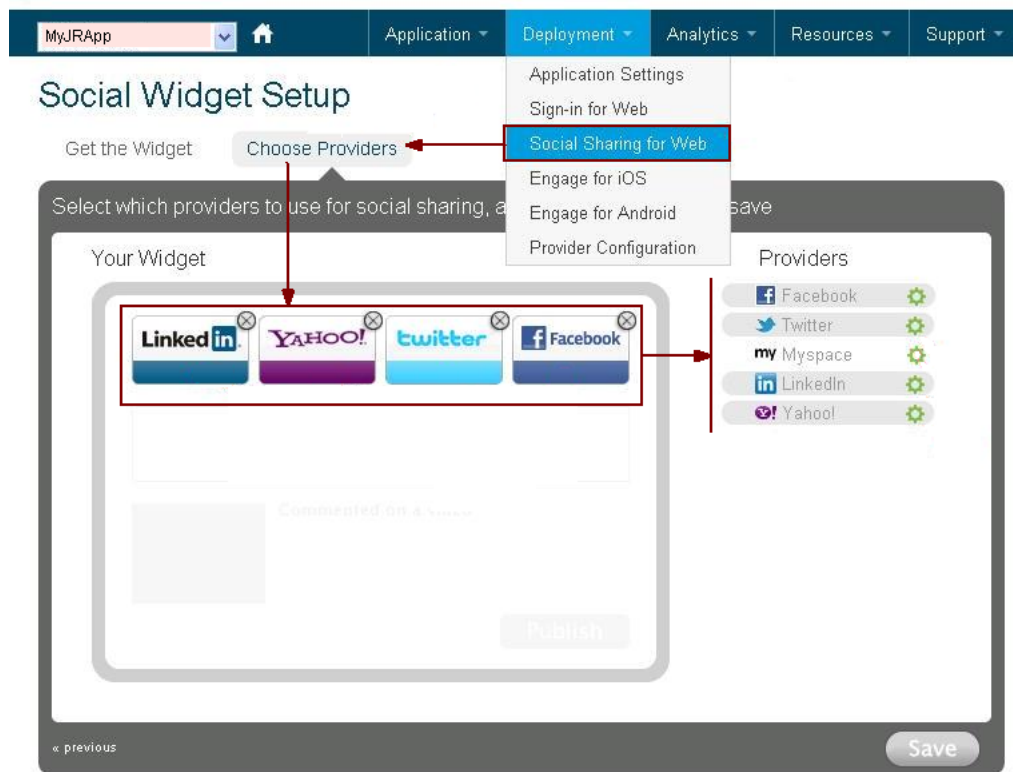
2.4.2 Configure Janrain Application's Authentication Settings on the Community Application

After configuring the Janrain application, configure Janrain authentication settings on the production Community application.

1. In the application server on the production system, navigate to the `cos.war/WEB-INF/classes` directory. For example, if you are using Tomcat, go to `<App_Server_Home>\webapps\cos\WEB-INF\classes`.

2. Open the `setup_auth.properties` file, then scroll down to the "Janrain Application Settings" section.

3. Enable Janrain support by setting the `widgets.external_auth.janrain.attrs.enabled` property to `true`.

4. For `widgets.external_auth.janrain.attrs.app_domain`, `widgets.external_auth.janrain.attrs.app_id`, and `widgets.external_auth.janrain.attrs.app_secret` configuration properties, enter the application domain, application ID, and API key you copied earlier.

The "Janrain Application Settings" section should look like this:

```
## Janrain Application Settings
#######################################
#
# Enabling Janrain support
# Default is "false"
#
widgets.external_auth.janrain.attrs.enabled=true
#
```

```
# Jainrain application domain
#
widgets.external_auth.janrain.attrs.app_domain=http://fw.rpxnow.com/
#
# Jainrain application id
#
widgets.external_auth.janrain.attrs.app_id=gajo1d134dfk3tgcoien
#
# Jainrain application secret
#
widgets.external_auth.janrain.attrs.app_secret=5asdf234jasdfk531
```

5. After saving the configuration file, restart the application server.

6. Open the Community interface, then go to **Login Bar** > **Configure**.

7. Select **Enable integration with Janrain platform** (Figure 2–17), then click **Save**. In addition, you can select the **Social sharing of comments and reviews via dedicated Janrain widget** check box if you configured the sharing option on the Janrain site.

> **Note:** Download the Janrain XD file to your local drive.
>
> To enable cross-domain communication, upload this file to the web site on which the Community application widgets will be deployed, and specify the file's new path in the "Janrain XD file URL" field. For instance, if the full URL of this file on the site is `http://acme.com/rpx_xdcomm.html`, then the value in the "Janrain XD file URL" field should be `/rpx_xdcomm.html`.

Figure 2–17   Janrain Support Enabled

8. Restart the management and production environments.

9. On the web site on which the Community application widgets are deployed, verify that the configured identity providers are displayed in the Community widget's "Login" screen, as shown in Figure 2–18.

Figure 2–18   Login Screen with Identity Provider Links
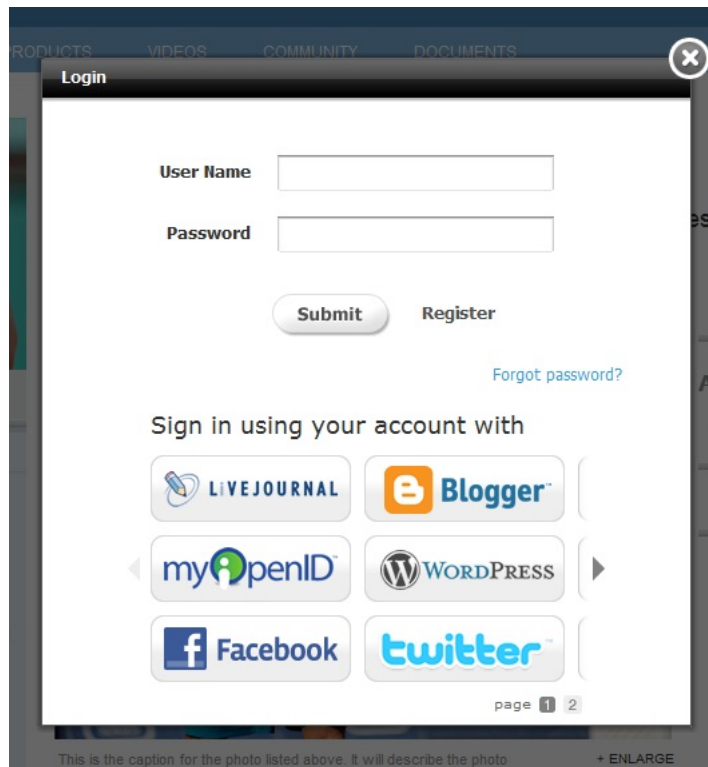
Note: If your Community application is running on Oracle WebLogic Server or Tomcat application server, then the procedures described in Section 2.4.1, "Create a Janrain Application for the Community Application" and Section 2.4.2, "Configure Janrain Application's Authentication Settings on the Community Application" are all you require to integrate the Community application with Janrain. However, on a WebSphere Application Server instance, you must also perform the export/import procedures described in Section 2.5, "Enabling Social Networking Services on WebSphere Application Server" to enable communication between the Community application and this social networking service.

2.5 Enabling Social Networking Services on WebSphere Application Server

If you integrated the Community application with Facebook, Twitter, or Janrain on a WAS instance, then you must also perform the procedures described in this section to enable communication between the Community application and these social networking sites.

This section includes the following:

- Section 2.5.1, "Export Security Certificate from Facebook"

- Section 2.5.2, "Export Security Certificate from Twitter"

- Section 2.5.3, "Export Security Certificate From Janrain"

- Section 2.5.4, "Import Security Certificates into WAS"

2.5.1 Export Security Certificate from Facebook

If you enabled native social integration and the Facebook login option in your Community application (for reference, see Figure 2–17), export the Facebook certificate. You will need this certificate to enable Facebook on WAS.

> **Tip:** The native login can be disabled via the Community interface > **Login Bar** > **Configure**. If you disable this feature, then the "Login" screen displayed on the web site will require visitors' local credentials (as used on the web site).

1. To get the Facebook security certificate, open this URL in FireFox: https://developers.facebook.com/docs/reference/api

2. From the "Tools" menu, choose **Page Info**.

3. On the "Security" tab, click **View Certificate** (Figure 2–19) to display the "Certificate Viewer" dialog box.

Figure 2–19   Facebook Security Certificate
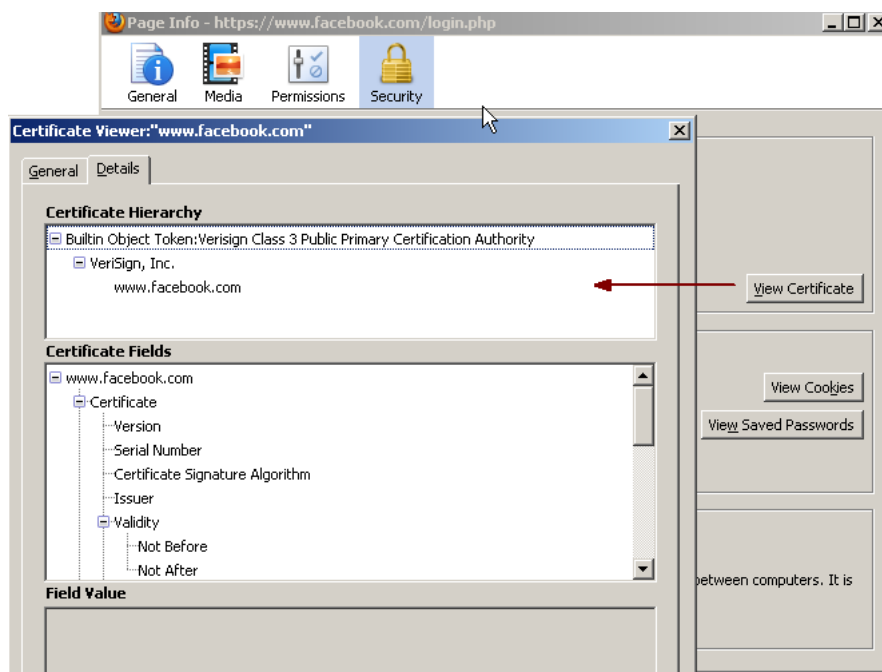
4. On the "Details" tab, click **Export**.

5. From the "Save in" dropdown list, choose a directory in which you want to save this certificate.

6. In the "Save Certificate to File" dialog box, choose **X.509 Certificate (DER)** from the "Save as type" field.

7. Click **Save**.

> **Note:** Facebook additionally requires the Entrust.net secure server certificate, which you can download from your Internet Explorer by choosing **Internet Options** > **Content** > **Certificates** > **Entrust.net Secure Server Certification Authority**.

2.5.2 Export Security Certificate from Twitter

If you enabled native social integration and the Twitter login option in your Community application (for reference, see Figure 2–17), export the Twitter certificate.

> **Tip:** The native login can be disabled via the Community interface > **Login Bar** > **Configure**. If you disable this feature, then the "Login" screen displayed on the web site will require visitors' local credentials (as used on the web site).

1. To get the Twitter security certificate, open this URL in FireFox:
   https://api.twitter.com

   The "Secure Connection Failed" dialog box is displayed.

2. From the "Tools" menu, choose **Page Info**.

3. On the "Security" tab, click **View Certificate**.

   The "Certificate Viewer" dialog box (Figure 2–20) is displayed.

Figure 2–20   Twitter Security Certificate
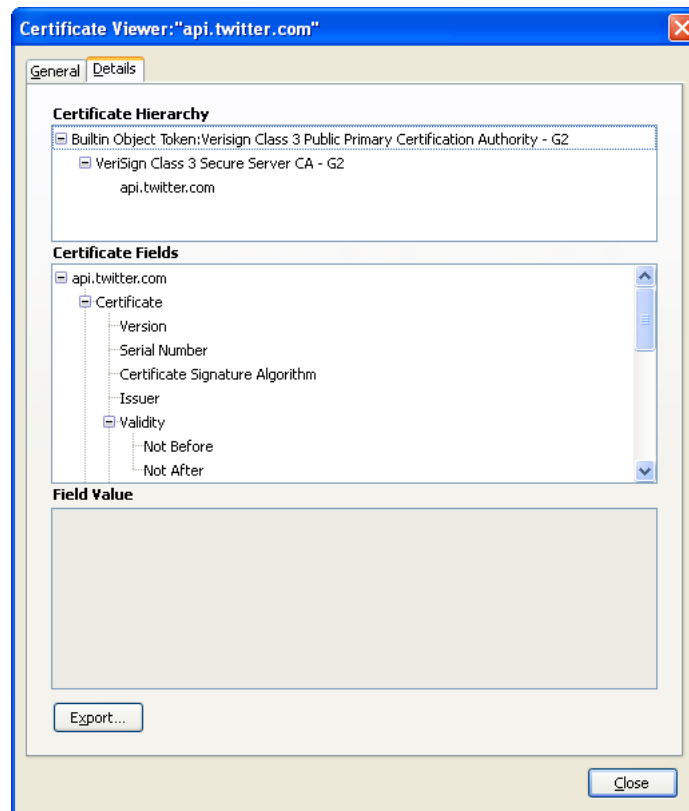
4. On the "Details" tab, click **Export**.

   The "Save Certificate to File" dialog box is displayed.

5. From the "Save in" dropdown list, choose the directory in which you want to save this certificate.

6. From the "Save as type" field, choose **X.509 Certificate (DER)**, as shown in Figure 2–21.

Figure 2–21   Save Certificate to File
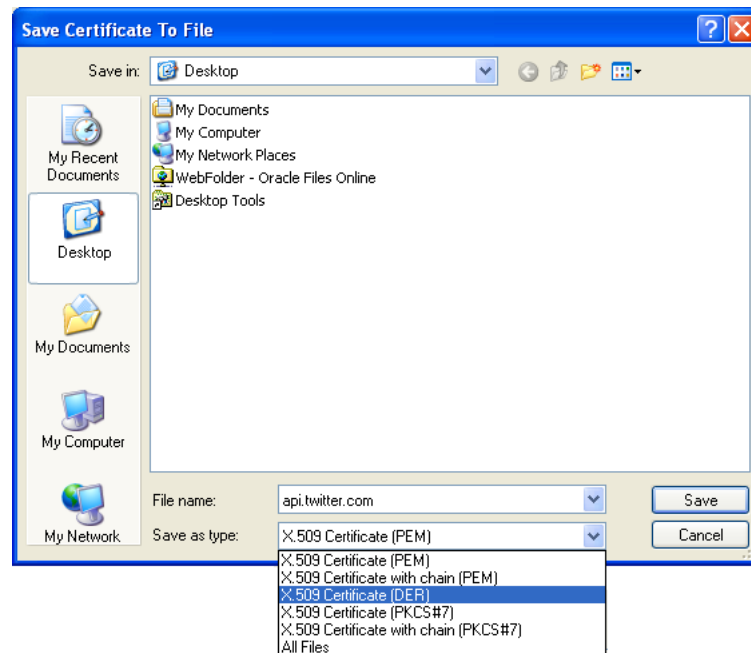
7.   Click **Save**.

2.5.3 Export Security Certificate From Janrain

To export the Janrain security certificate:

1.   Open this URL in FireFox: `https://rpxnow.com/`

2.   From the "Tools" menu, choose **Page Info**.

3.   On the "Security" tab, click **View Certificate** to display the "Certificate Viewer" dialog box (Figure 2–22).

Figure 2–22   Janrain Security Certificate
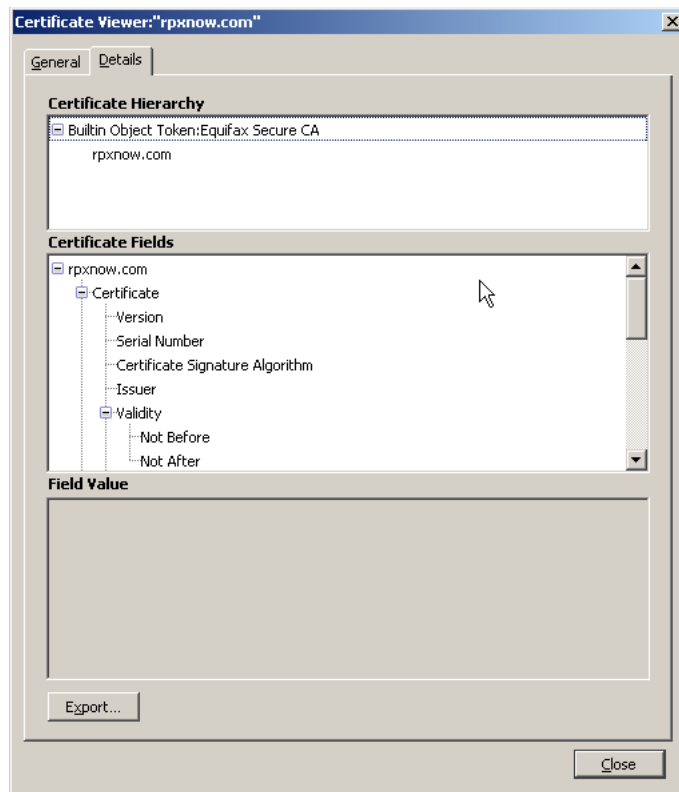
4. On the "Details" tab, click **Export**.

5. From the "Save in" dropdown list, choose a directory in which you want to save this certificate.

6. In the "Save Certificate to File" dialog box, choose **X.509 Certificate (DER)** from the "Save as type" field.

7. Click **Save**.

2.5.4 Import Security Certificates into WAS

To import Facebook, Twitter, or Janrain security certificates into WAS:

1. Log in to the WAS administrative console.

2. Expand **Security**, then click **SSL certificate and key management**.

3. Under "Configuration settings", click **Manage endpoint security configurations**, as shown in Figure 2–23.

Figure 2–23   Configuration Settings

4. Under the "Local Topology" tab, expand **Outbound**, then select the appropriate outbound configuration to get to the (cell).

5. Under "Related Items" on the right side, click **Key stores and certificates**.

6. Under "Preferences", select the **CellDefaultTrustStore** key store, as shown in Figure 2–24.

Figure 2–24   Preferences - CellDefaultTrustStore

7. Under "Additional Properties" on the right side, click **Signer certificates**.

8. Under "Preferences", click **Add** (Figure 2–25).

Figure 2–25   Preferences - Add

9. In the "Alias" field, enter `graph.facebook.com_cert`, `api.twitter.com_cert`, or `rpxnow.com_cert` depending on the certificate you are adding, as shown in Figure 2–26.

10. In the "File name" field, enter the path to the saved Facebook/Twitter certificates, as shown in Figure 2–26.

11. From "Data type", choose **Binary DER data**, as shown in Figure 2–26.

Figure 2–26   SSL Certificate and Key Management - Data Type

12. Click **Apply**.

3

Customizing the Community Application's Functionality

The Community application provides a number of customizability and extension points, such as CSS, templates, and Search Engine Optimization (SEO), which you can use to alter the appearance and functionality of any Community widget. However, using customizability and extension points may not always meet particular customers' project requirements because of certain web site design or usage related strategies applied to the widgets. In such scenarios, consider working with WebCenter Sites data structures directly and rendering them separately in a custom way in page templates.

This chapter describes WebCenter Sites data structures in details. It includes the following sections:

- Section 3.1, "Overview of Community Data Model"
- Section 3.2, "Customizing CSS and Widget Templates"
- Section 3.3, "Creating a Custom Word Filter"
- Section 3.4, "Creating a CAPTCHA Generator"

3.1 Overview of Community Data Model

The Community application uses WebCenter Sites as its data repository and communicates with it through REST APIs provided by Web Experience Management (WEM) Framework. Storing the Community application's data structures in the WebCenter Sites repository leverages the WebCenter Sites asset model and caching system.

The object data model of the Community application consists of the following categories and their objects:

- User-Generated Content
  - Comments
    - Object: CommentFeed
    - Object: CommentRecord
  - Reviews
    - Object: ReviewFeed
    - Object: ReviewRecord
  - Ratings
    - Object: RatingFeed

- Object: RatingRecord
    - Polls
        - Object: Poll
    - Common Infrastructure: Topics
        - Object: Topic
- Visitors
    - Object: User
    - Object: UserIdentity
    - Object: UserLink

Figure 3–1 provides a graphical overview of these categories and objects. The "User-Generated Content" section shows that business objects are related to Community widgets, such as comments and reviews. Each user-generated content (UGC) entry—if posted by a registered user—has a set of user objects (see the "Visitors" section) associated with it.

Figure 3–1   Categories and Object Layout
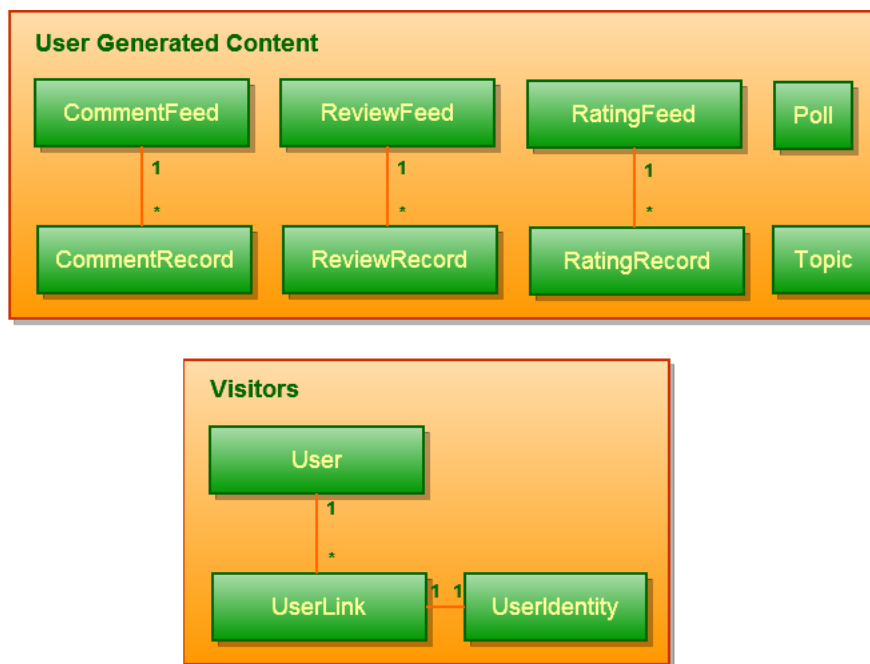
Objects shown in Figure 3–1 are stored in the WebCenter Sites database tables, as specified in Table 3–1.

Table 3–1   Database Table Names of Objects

Object	Database Table Name
CommentFeed	cos_comment_feed
CommentRecord	cos_comment_record
ReviewFeed	cos_review_feed
ReviewRecord	cos_review_record

Table 3–1   (Cont.) Database Table Names of Objects

Object	Database Table Name
RatingFeed	cos_rating_feed
RatingRecord	cos_rating_record
Polls	cos_poll
Topics	cos_topic
User	cos_user
UserIdentity	cos_user_id
UserLink	cos_user_link

3.1.1 Comments

The following types of primary objects are associated with comments:

■ CommentFeed

■ CommentRecord

3.1.1.1 CommentFeed

This object represents the computed summary information about the web page on which the widget is deployed. The CommentFeed object is created in the database when the page on which the Comments widget is deployed is accessed for the first time.

To determine the connection between comments and the page on which they should be displayed, the Community application uses the web page URL, which is the application-generated key in the widget deployment tag, or "Resource ID", which administrators can specify during widget deployment. If the web page URL is available as the key, then the hash generated from the page URL is recorded in the CommentFeed object. If the resource ID is specified, it is recorded in the CommentFeed object. The CommentFeed object stores comments, and it has a one-to-many association with the CommentRecord objects that represent comments.

The CommentFeed objects are stored as a `cos_comment_feed` basic asset in WebCenter Sites and correspondingly as a `cos_comment_feed` table in the database. Table 3–2 describes the structure of this asset.

Table 3–2   Structure of the "cos_comment_feed" AssetType

Property	Type	Description
id	BIGINT	Unique asset identifier or primary key of AssetType.

Table 3–2   (Cont.) Structure of the "cos_comment_feed" AssetType

Property	Type	Description
cos_resource_id	VARCHAR(256)	Identifier of the discussion. It is a logical link that the Community application uses to associate a list of comments with the page on which visitors posted comments. This link is established during widget deployment by specifying the value of the "Resource ID" field on the deployment page or the resource_id attribute in the widget code snippet. If it is not specified, the Community application uses the hash of the page URL as a resource ID. It is recommended that you come up with your own strategy of resource_id generation. For example, to make it easier to find this object in the database, you can use the page ID as the resource ID.
cos_url	VARCHAR(256)	URL that hosts the comment discussion thread.
cos_date_created	DATE	Date when the feed object was created.
cos_date_modified	DATE	Latest modification date of a feed.
cos_modified_by	VARCHAR(256)	ID of the user who last modified the feed.
cos_owner	VARCHAR(256)	ID of the user who created the feed object. (The feed object is created during the first page load when a widget is just deployed. Typically it is done using a guest session, so -1 is a typical value.)
cos_approved_count	BIGINT	Number of comments that successfully passed moderation and are displayed to visitors.
cos_pending_count	BIGINT	Number of comments that have not passed moderation yet.
cos_ext_type	VARCHAR(256)	ID of the content category. It describes the type of content to which comments are attached: file, article, blog, and so on. (Specified as the "Resource Type" parameter on the deployment page or the resource_type attribute in the widget deployment code snippet.)
cos_resource_title	VARCHAR(256)	Title of the page on which the Comments widget is deployed. By default, the value of the window title is recorded here. Page designers may customize this value by specifying the resource_title attribute in the deployment code snippet.

3.1.1.2 CommentRecord

This object represents a comment posted by a visitor on a site page. It is logically linked to the CommentFeed object through a many-to-one relationship via the cos_root_id field. In other words, many CommentRecord objects can be linked to one CommentFeed object.

Table 3–3   Structure of the cos_comment_record assetType

Property	Type	Description
id	BIGINT	Identifier of a comment.
cos_text	VARCHAR(4000)	Content of a posted comment. Its VARCHAR(4000) data type is translated into the most effective type of storage (for example, CLOB), depending on the database on which WebCenter Sites is installed.
cos_root_id	BIGINT	Association to the CommentFeed object representing the page on which the conversation is happening. The value of the corresponding cos_comment_feed.id item is stored here. Use this column to query all the comments posted on a page.
cos_state_value	VARCHAR(450)	State of a comment reflecting the stage of comment moderation. If manual moderation is enabled, the value may be: ■ pending.new (newly posted) ■ pending.modified (manual moderation) If a comment has either passed moderation or is auto-published, the value will be: ■ approved.all If a comment has not passed through either automatic filters or moderator, then the values are: ■ inappropriate.robotdetected ■ inappropriate.humandetected To simplify navigation for moderators, the Community interface includes filters based on these categories.
cos_owner	BIGINT	ID of the visitor who posted the comment, or -1 if it is a guest entry.
cos_owner_ip	VARCHAR(256)	Client's IP address from which the comment was posted.
cos_guest_name	VARCHAR(450)	Display name of a registered user or a guest who posted the comment entry. This field is used in the Community interface to display and sort comments by author names without posting extra requests to database.
cos_guest_email	VARCHAR(450)	If guest (unauthenticated visitors) posts are enabled in commenting permissions, and visitors are required to specify their email ID, then the email ID is saved in this column.
cos_level	BIGINT	Level of a comment in the comment thread, if replies to comments are enabled. The bigger the number, the deeper the comment is posted in the tree. The initial value is 1.

Table 3–3   (Cont.) Structure of the cos_comment_record assetType

Property	Type	Description
cos_parentid	BIGINT	ID of the immediate parent comment to which a reply is posted.
cos_parent0id - cos_parent9id	BIGINT	Chain of comment parents in the hierarchy. The path from the leaf to the root.
cos_flagged	VARCHAR(32)	`true` or `false` based on whether the comment has been flagged and reported by other visitors.
cos_flagged_count	BIGINT	The number of times a comment is flagged.
cos_record_rank	VARCHAR(450)	Counts of helpfulness reports ("Yes" and "No") made on a comment, separated by comma, starting with zero. The initial value is `0,0`.
cos_record_rank_calculated	INTEGER	Pre-calculated rank of a comment according to helpfulness reported by other visitors. This is calculated by subtracting the count of "No" from the count of "Yes".
cos_date_created	DATE	Date when the comment was posted.
cos_date_modified	DATE	Latest modification date of a comment (registered users are allowed to modify their comments, or moderator may modify a comment.).
cos_reply_count	BIGINT	If the discussion thread is enabled and replies can be posted to comments, then the number of replies to the current comment are recorded here. This count is not recursive, and therefore, only immediate child comments are considered (the children of children are excluded).
cos_thread_order	VARCHAR(450)	String generated in a special format. It enables ordering of comments by thread hierarchy using a simple WEM REST query and alphabetical ordering. The generation strategy is as follows: ■ If it is a root level comment (that is, it has not been posted as a reply to another comment), the value of the `cos_date_created` field in the hexadecimal format is recorded. ■ If a comment has a parent comment, the concatenation value of parent's `cos_comment_record.cos_thread_order` field and the `underscore "_"` field, as well as its `cos_date_created` field in the hexadecimal format.

3.1.2 Reviews

The following types of primary objects are associated with reviews. These objects are similar to those of comments:

■ ReviewFeed

■ ReviewRecord

3.1.2.1 ReviewFeed

This object represents a list of reviews posted on a site page. The ReviewFeed object is very similar to the CommentFeed object described in Section 3.1.1.1, "CommentFeed." The Community application uses the same code infrastructure to handle these objects. However, to simplify maintenance on the database schema level, the data for CommentFeed and ReviewFeed is stored in two separate tables. The only difference between CommentFeed and ReviewFeed is that the latter includes additional information pertaining to the average rank calculated across all posted reviews.

The ReviewFeed object is created in the database when the page on which the Reviews widget is deployed is accessed for the first time.

To determine the connection between reviews and the page on which they should be displayed, the Community application uses the web page URL, which is the application-generated key in the widget deployment tag, or "Resource ID", which administrators can specify during widget deployment. If the web page URL is available as the key, then the hash generated from the page URL is recorded in the ReviewFeed object. If the resource ID is specified, it is recorded in the ReviewFeed object. The ReviewFeed object stores reviews, and it has a one-to-many association with the ReviewRecord objects that represent reviews.

The review feed objects are stored as a `cos_review_feed` basic asset in WebCenter Sites and correspondingly as a `cos_review_feed` table in the database. Table 3–4 describes the structure of this asset.

Table 3–4   Structure of cos_review_feed AssetType

Property	Type	Description
id	BIGINT	Unique asset identifier or primary key of the AssetType.
cos_resource_id	VARCHAR(256)	Identifier of a discussion. It is a logical link that the Community application uses to associate a list of reviews with the page on which the reviewed topic is posted. This link is established during widget deployment by specifying the value of the "Resource ID" field on the deployment page or the `resource_id` attribute in the widget code snippet. If it is not specified, the Community application uses the hash of the page URL as a resource ID. It is recommended that you devise your own strategy of `resource_id` generation. For example, to make it easier to find this object in the database, you can use the page ID as the resource ID.
cos_url	VARCHAR(256)	URL hosting the discussion to which reviews are being posted.
cos_date_created	DATE	Date when the feed object was created.
cos_date_modified	DATE	Latest modification date of a feed.
cos_modified_by	VARCHAR(256)	ID of the user who last modified the feed.
cos_owner	VARCHAR(256)	ID of the user who created the feed object. (The feed object is created during the first page load when a widget is just deployed. Typically it is done using a guest session, so -1 is a typical value.)

Table 3–4 (Cont.) Structure of cos_review_feed AssetType

Property	Type	Description
cos_approved_count	BIGINT	Number of reviews that successfully passed moderation and are displayed to visitors.
cos_pending_count	BIGINT	Number of reviews that have not pass moderation yet.
cos_rank	FLOAT	Calculated as a mean, the average of all ranks that have been given to the reviews posted on a page. For example, if two reviews were posted on a page, out of which one has been given three stars and another five stars, then the value in this field will be 3+5/2 = 4.
cos_rank_ precalculation	VARCHAR(450)	Comma separated list of counts of ranks posted. This list contains five items in total. Each position (index) in the list holds the count of the corresponding number of stars given to a review. For example, if there are three reviews, out of which one has been given three stars and the others five stars, 0-s will be given for missing positions, and therefore, the list will contain 0,0,1,0,2. Note the counts of reviews posted at corresponding positions. From concrete position, you can know the number of votes/stars.
cos_thumbs_up_rank	VARCHAR(450)	If the review type is set to thumbs up/down, the votes are recorded in this field. This is a denormalized field that holds a count of thumbs up and thumbs down separated by comma. For example, if out of three reviews, two have got thumb up and one thumb down, then the value of this field will be 2,1,.
cos_ext_type	VARCHAR(256)	ID of the content category describing the type of content to which the reviews are attached, for example, file, article, blog, and so on. (Specified as the "Resource Type" parameter on the deployment page or the resource_type attribute in the widget deployment code snippet.)
cos_resource_title	VARCHAR(256)	Title of the page on which the Reviews widget is deployed. By default, the value of the window title is recorded here. Page designers may customize this value by specifying the resource_title attribute in the deployment code snippet.

3.1.2.2 ReviewRecord

This object represents a review posted by a visitor on a site page. It is logically linked to the ReviewFeed object through a many-to-one relationship via the cos_root_id field.

Table 3–5 Structure of the cos_review_record AssetType

Property	Type	Description
id	BIGINT	Identifier of a review.

Table 3–5　(Cont.) Structure of the cos_review_record AssetType

Property	Type	Description
cos_text	VARCHAR(4000)	Content of a review posted. Its VARCHAR(4000) data type is translated into the most effective type of storage (for example, CLOB), depending on the database on which WebCenter Sites is installed.
cos_title	VARCHAR(450)	Title of a review posted.
cos_rank	FLOAT	Number of stars (rank) given to a review by a visitor. If the rating type is thumbs up/down, then 5 is the value for a thumbs up and 1 for a thumbs down.
cos_thumbs_up	INTEGER	If the rating type is configured to be thumbs up/down, the rank value stored here is: 1 for thumbs up and -1 for thumbs down.
cos_rating_type	INTEGER	Type of a given review rank. Possible value codes are: 0 - Stars 1 - Thumbs up/down
cos_root_id	BIGINT	Association to the ReviewFeed object representing the page on which reviews are submitted. The value of the corresponding cos_review_feed.id item is stored here. Use this column to query all the reviews posted on a page.
cos_state_value	VARCHAR(450)	State of a review reflecting the stage of content moderation. If manual moderation is enabled, the value may be: ■　pending.new (newly posted) ■　pending.modified (manual moderation happened) If a review has either passed moderation or was auto-published, the value will be: ■　approved.all If a review has not passed through either automatic filters or moderator, values will be: ■　inappropriate.robotdetected ■　inappropriate.humandetected To simplify navigation for moderators, the Community interface includes filters based on these categories.
cos_owner	BIGINT	ID of the visitor who posted the review, or -1 if it is a guest entry.
cos_owner_ip	VARCHAR(256)	Client's IP address from which the review was posted.

Table 3–5   (Cont.) Structure of the cos_review_record AssetType

Property	Type	Description
cos_guest_name	VARCHAR(450)	Display name of a registered user or a guest who posted the review entry. This field is used to display and sort reviews by author names in the Community interface without making extra requests to database.
cos_guest_email	VARCHAR(450)	If guest (unauthenticated visitors) posts are enabled in reviewing permissions and visitors are required to specify their email ID, the email ID value is saved in this column.
cos_flagged	VARCHAR(32)	true or false based on whether a review has been flagged and reported by other visitors.
cos_flagged_count	BIGINT	The number of times a review has been flagged by others.
cos_record_rank	VARCHAR(450)	Counts of helpfulness reports ("Yes" and "No") made on a review, separated by comma, starting with zero. The initial value is 0,0.
cos_record_rank_ calculated	INTEGER	Pre-calculated rank of a review according to helpfulness reported by other visitors. It is the count of "No" subtracted from the count of "Yes".
cos_date_created	DATE	Date when the review was posted.
cos_date_modified	DATE	Latest modification date of a review. (Only registered users are allowed to modify their reviews. Moderator can also modify a review, if needed.)

3.1.3 Ratings

This object contains the list of ratings that visitors give to topics or any posts on a page. This is very similar to the ReviewFeed object.

The Ratings object is created in the database when a web page is accessed for the first time after the deployment of the Ratings widget on a web site.

To determine the connection between ratings and the page on which they should be displayed, the Community application uses the web page URL, which is the application-generated key in the widget deployment tag, or "Resource ID", which administrators can specify during widget deployment. If the web page URL is available as the key, then the hash generated from the page URL is recorded in the RatingFeed object. If the resource ID is specified, it is recorded in the RatingFeed object. The RatingFeed object stores ratings, and it has a one-to-many association with the RatingRecord objects that represent ratings.

The rating feed objects are stored as the *cos_rating_feed* basic asset in the WebCenter Sites repository and correspondingly as the *cos_rating_feed* table in the database.

The following types of primary objects are associated with ratings:

- RatingFeed
- RatingRecord

Table 3–6 and Table 3–7 describe the structure of the cos_rating_feed assetType.

3.1.3.1 RatingFeed

Table 3–6   Structure of the cos_rating_feed AssetType

Property	Type	Description
id	BIGINT	Identifier of an object which holds a list of ratings posted on a page.
cos_resource_id	VARCHAR(256)	Identifier of the ratings on a particular web page.
		It is a logical link which enables the Community application to associate a list of ratings with the relevant page. This link is established during widget deployment by specifying the value of the "Resource ID" field on the deployment page or the resource_id attribute in the widget code snippet. This value can be generated or filled manually. If this value is not specified, the Community application uses the hash of page URL as a resource ID.
		It is recommended that you create your own strategy to generate resource_id. For example, to make the database search of this object easier, you can use the page ID as the resource ID.
cos_url	VARCHAR(256)	URL of the web page on which visitors leave their ratings.
cos_date_created	DATE	Date when the feed object is created.
cos_date_modified	DATE	Last modified date of a feed.
cos_owner	VARCHAR(256)	ID of the user who created the feed object initially. (This happens during the first page load when a widget is just deployed. Typically, the feed object is created via a guest session, so -1 is a typical value.)
cos_approved_count	BIGINT	Number of ratings that successfully passed moderation and are counted in the general rating calculation.
cos_pending_count	BIGINT	Number of ratings that have not passed moderation yet.
cos_stars_rank	FLOAT	Average (mean) of all star type ratings posted by visitors. For example, if out of the two ratings, one is a three-star rank and another is a five-star rank, then the value is calculated as 3+5/2 = 4.
cos_stars_rank_ precalculation	VARCHAR(450)	Comma separated list of counts of ratings. This list contains five items in total. Each position (index) in the list holds the count of the corresponding number of stars given. For example, if out of the three ratings given, one is a three-star and the other two are five-star, then 0-s in this list signify missing positions. So the list will include: 0,0,1,0,2. Note the counts of ratings left at corresponding positions. From concrete positions you can know the number of votes/stars.

Table 3–6   (Cont.) Structure of the cos_rating_feed AssetType

Property	Type	Description
cos_thumbs_up_rank	VARCHAR(450)	If the rating type is set to thumbs up/down, then the ratings are recorded in this field. This is a denormalized field to hold a comma separated count of thumbs up and thumbs down. For example, if out of the three ratings, the two are thumbs up and one is thumbs down, then the value stored will be equal to 2,1.
cos_likeit_count	BIGINT	If the like it rating type is deployed, then the number of likes are recorded in this field.
cos_recommend_ count	BIGINT	If the recommend rating type is deployed, then the number of recommendations is recorded in this field.
cos_ext_type	VARCHAR(256)	ID of a content category. It describes the type of content to which the ratings are attached: file, article, blog, and so on. These are specified as the value of the "Resource Type" parameter on the deployment page or the resource_type attribute in the widget deployment code snippet.
cos_resource_title	VARCHAR(256)	Title of the page on which the Ratings widget is deployed. By default the value of the window title is recorded as the page title. Page designers can customize this value by specifying the resource_title attribute in the deployment code snippet.

3.1.3.2 RatingRecord

This object represents a rating posted by a visitor on a site page. It is logically linked to the RatingFeed object via the "cos_root_id" field. This object is based on the many-to-one relationship model.

Table 3–7   Structure of the cos_rating_record AssetType

Property	Type	Description
id	BIGINT	Identifier of a rating.
cos_thumbs_up	INTEGER	If the rating type is configured to be thumbs up/down, then the rank value recorded here is: 1 if it is thumbs up and -1 if it is thumbs down.
cos_rating_type_value	INTEGER	Type of rating rank. Possible value codes are: 0 - stars, 1 - thumbs up/down, 2 - like it, 3 - recommend.
cos_root_id	BIGINT	Association to the RatingFeed object representing the page on which ratings are submitted. The value of the corresponding cos_rating_feed.id item is recorded here. Use this column to query all the ratings posted on a page.

Table 3–7 (Cont.) Structure of the cos_rating_record AssetType

Property	Type	Description
cos_state_value	VARCHAR(450)	State of ratings reflecting the stage of content moderation.
		If manual moderation is enabled, the value may be pending.new (newly posted) or pending.modified (manual moderation).
		If a rating has either passed moderation or was auto-published, the value will be approved.all.
		If a rating has not pass through either automatic filters or moderator, then the value will be inappropriate.robotdetected or inappropriate.humandetected.
		To simplify navigation for moderators, the Community interface provides filters for these categories on the right-side panel.
cos_owner	BIGINT	ID of the visitor who posted the rating. Or -1 if it is a guest entry.
cos_owner_ip	VARCHAR(256)	Client's IP address from which the rating is posted.
cos_guest_name	VARCHAR(450)	Display name of a registered user or a guest who gave the rating. This field is used to display and sort ratings by author names in the Community interface without making extra requests to the database.

3.1.4 Polls

The Polls functionality is represented in a single table in the database. When an administrator creates a poll, the corresponding row is inserted into this table (Table 3–8).

Table 3–8 Structure of the cos_poll AssetType

Property	Type	Description
id	BIGINT	Identifier of a poll instance.
cos_uid	VARCHAR(256)	Unique string identifier of the poll instance. It is embedded into the deployment code snippet along with the regular ID so that if the integer ID is lost during data migration, the system can look up this poll by UID, thus preventing existing deployments from breaking.
cos_chart_type	INTEGER	Type of the chart in which poll results are to be displayed. Possible values are: ■ 0 - Pie chart ■ 1 - Bar chart ■ 2 - Flat Results (percents are shown for options)

Table 3–8   (Cont.) Structure of the cos_poll AssetType

Property	Type	Description
cos_theme	INTEGER	ID of the theme to be applied to Poll on a web site. Possible values are: ■ 0 - Basic ■ 1 - Advanced ■ 2 - No design ■ 3 - Open design
cos_start_date	DATE	Date when the poll is started and opened for votes. The start date of the poll voting campaign.
cos_finish_date	DATE	Date when the poll is closed and no more voting is allowed. The end date of poll voting campaign.
cos_result_required	VARCHAR(32)	Boolean value (true or false) that specifies whether to show results after voting.
cos_results_view	INTEGER	Determines how poll results are shown: ■ 0 - In a pop-up ■ 1 - In place ■ Inside the poll widget
cos_results_width	INTEGER	Widget for poll results pop-up in pixels.
cos_title	VARCHAR(450)	Title of the poll.
cos_question	VARCHAR(450)	Main poll question.
cos_options	VARCHAR(4000)	List of poll options to be available for voting in the JSON format. The metadata includes ID of the vote used by the Community application internally, the option title, color, and the number of votes left. For example: `[{"id":"n0","count":1,"color":"#1751a7","value":"Avatar"},{"id":"n1","count":0,"color":"#8aa717","value":"Matrix"}]`
cos_thankyou_note	VARCHAR(450)	Message to be shown to visitors after their vote.
cos_disclaimer_required	VARCHAR(32)	Boolean value (true or false) that specifies whether to show the disclaimer text for poll or not.
cos_disclaimer	VARCHAR(450)	Disclaimer text to be shown at the bottom of the Poll widget.
cos_votes	INTEGER	Total number of votes on the current poll campaign.

3.1.5 Topics

The Topics functionality allows to gather and precalculate statistics, such as review counts, rating values, and so on, across the Community application widgets so that the collected information can be easily and efficiently queried later. The precalculation and optimization of visitors' feedback is important because this type of content may appear on the home page of a site or on the visitor's dashboard that lists most discussed/reviewed/rated articles.

Web pages on which the Community application widgets are deployed and activities performed on those pages become the primary focus of the Topics functionality. Therefore, statistics are aggregated across tables such as CommentFeed, ReviewFeed, and RatingFeed.

Table 3–9   Structure of the cos_topic AssetType

Property	Type	Description
id	BIGINT	Identifier of a topic instance.
cos_url	VARCHAR(450)	URL of the page on which widgets are deployed.
cos_title	VARCHAR(450)	HTML title of the page with which this topic is associated.
cos_resource_id	VARCHAR(450)	Resource ID of the page that links the UGC content to the page on which widgets are deployed.
cos_date_created	DATE	Date when the topic is created. That is, when the page on which widgets are deployed is accessed for the first time using a browser.
cos_comments_feed_id	BIGINT	Relation between the corresponding CommentFeed object and the CommentFeed.id field. This is a source object from which the statistics are aggregated.
cos_comment_count	BIGINT	Number of comments posted on the page with which a particular topic is associated.
cos_comments_resource_type	VARCHAR(450)	Content category (such as a blog or article) assigned to the Comments widget deployed on the page. It might be either the default category value or the customized category uploaded on the Appearance settings page.
cos_comments_date_modified	DATE	Date when the CommentFeed object is last updated with statistic values (the number of comments posted, and so on).
cos_reviews_feed_id	BIGINT	Relation to the ReviewFeed object with which a topic is associated.
cos_review_count	BIGINT	Number of reviews posted on a page with which a topic is associated.
cos_reviews_resource_type	VARCHAR(450)	Content category (such as a blog or article) assigned to the Reviews widget deployed on a page. It might be either a default category or the customized category uploaded on the Appearance settings page.
cos_reviews_date_modified	DATE	Date when the ReviewFeed object is last updated with statistic values (the number of reviews posted, ranks, and so on).
cos_ratings_feed_id	BIGINT	Relation to the RatingFeed object with which a particular topic is associated.
cos_rating_count	BIGINT	Number of ratings posted on a page with which this topic is associated.
cos_ratings_resource_type	VARCHAR(450)	Content category (such as blog, article, and so on) assigned to the Ratings widget deployed on a page. It might be either a default category or the customized category uploaded on the Appearance settings page.

Table 3–9 (Cont.) Structure of the cos_topic AssetType

Property	Type	Description
cos_ratings_date_ modified	DATE	Date when the RatingFeed object is last updated with statistic values (the number of ratings posted, their ranks, and so on).
cos_rank	FLOAT	Rank of the current topic among other topics. The current ranking schema is based on the frequency statistics and is calculated as a sum of the following fields: cos_ review_count + cos_comment_count + cos_ rating_count.

3.1.6 Visitors

The following types of primary objects are associated with visitors:

- User

- UserIdentity

- UserLink

3.1.6.1 User

The User table represents a visitor profile maintained by the Community application, and it contains visitor-sensitive data such as display name, email, or avatar picture.

Table 3–10 Structure of the cos_user AssetType

Property	Type	Description
id	BIGINT	Identifier of a user profile.
cos_email	VARCHAR(450)	Email of a web site visitor. May not be populated if visitors logged in using Facebook, Twitter, or through Janrain. The visitors that registered locally may use it to recover their passwords when needed.
cos_display_name	VARCHAR(450)	Display name of a user. Either provided by a visitor during registration or taken from the social networking service using which the user logged in.
cos_profile_type_code	INTEGER	Type of the user profile. Possible values are: ■ 0 - Regular user or visitor ■ 1 - Editorial or management user (for example, content moderator) ■ 2 - System user or the user used by the system to open connections internally.

Table 3–10   (Cont.) Structure of the cos_user AssetType

Property	Type	Description
packed_identities	VARCHAR(4000)	Denormalized list of visitor identities in the JSON format. These identities are used during authentication, and therefore, they are linked to this profile.
		For example: [{"username":"john"}]
		The "username" field is related to a corresponding UserIdentity entry via the UserIdentity.cos_username field. For the visitor profile of the user who logged in using a social network service via Janrain, the user name will include the "jr:" prefix:
		[{"username":"jr:http://twitter.com/account/profile?user_id=3333333"}]
cos_picture_blob	BINARY	For a profile of a local user who registered locally and did not use a social networking service, the customized avatar is stored in this field.
cos_picture_url	VARCHAR(450)	For a profile of a user who used a social networking service to log in, the URL of the avatar picture is stored here.

3.1.6.2 UserIdentity

The two options for storing visitor credentials are LDAP and the WebCenter Sites database. The database option is set by default.

When the database is used, credentials of a user are stored in the UserIdentity table and associated with the corresponding visitor profile using the UserLink table.

Table 3–11   Structure of the cos_user_id AssetType

Property	Type	Description
id	BIGINT	Identifier of a user's credentials.
cos_username	VARCHAR(256)	User name used for authentication.
cos_email	VARCHAR(256)	Email of a user for the purpose of recovering user's password when needed.
cos_provider_id	VARCHAR(256)	Type of identity provider used for authenticating a visitor.
		Possible values are:
		■ none - if this is the identity of a system user who opened connections internally.
		■ ldap - if visitor authenticates via LDAP
		■ wem-db - if visitor authenticates via the Community application plug-in that uses a WebCenter Sites assetTypes.
cos_encryption_type	VARCHAR(256)	Encryption algorithm applied to secure the password of a user identity. The algorithm used in the current version of the system is blowfish.
cos_password	VARCHAR(256)	Encrypted value of a password used for authentication.

3.1.6.3 UserLink

Table 3–12 links UserIdentity to User so that a visitor's identities (that have credentials for authentication) are properly associated with the visitor's user profile. This is a denormalized table so that some of the fields from the UserIdentity table are cached here as well.

Table 3–12   Structure of the cos_user_link AssetType

Property	Type	Description
id	BIGINT	Identifier of a user link.
cos_username	VARCHAR(256)	User name from the UserIdentity object with which this link associates the User object.
cos_email	VARCHAR(256)	Email from the UserIdentity object with which this link associates the User object.
cos_provider_id	VARCHAR(256)	Identity provider code from the UserIdentity object with which this link associates the User object. Possible values are: - ext_auth - if visitor comes from social networks like Facebook or Twitter, or through Janrain. - none - if this is the identity of a system user who opens connections internally - ldap - if visitor authenticates via LDAP - wem-db - if visitor authenticates via the Community application plug-in which uses WebCenter Sites assets.
cos_account_id	BIGINT	Identifier of a User object with which this link associates the UserIdentity object.

3.2 Customizing CSS and Widget Templates

You can customize Community widgets in the following ways:

- **Color Schema and Skinning via CSS**. The CSS is customized when a widget's look-and-feel must match that of the web site on which the widget is deployed.

- **Redesign via Widget Templates.** Widget templates are customized when a widget requires significant changes, such as a new layout or enhancements to functionality.

The following sections describe how to perform these customizations:

- Section 3.2.1, "Customizing CSS: Color Schema and Skinning"

- Section 3.2.2, "Customizing a Widget Template"

3.2.1 Customizing CSS: Color Schema and Skinning

The general steps for customizing the CSS are:

1. Download the standard CSS skin of the widget.

2. Modify CSS styles for the interface elements that require customization.

3. Apply the customized CSS.

While the overall process is the same, customizing CSS styles for Comments and Reviews widgets is different from customizing other widgets.

This section includes the following:

- Section 3.2.1.1, "Customizing Comments and Reviews Widgets"
- Section 3.2.1.2, "Customizing Other Widgets"

3.2.1.1 Customizing Comments and Reviews Widgets

For Comments and Reviews, you can download the default CSS from the widget's "Appearance" page in the Community interface, modify it, and then upload it back. The Community application continues to host the CSS while it is being customized, so you can change it anytime you like.

To customize the Comments CSS:

1. Log in to the Community application as an administrator.

2. From the "Comments" menu, choose **Configure**, then **Appearance** to display the "Comments Appearance" page.

3. In the "General" section, click **Download the current CSS** next to the "Upload custom CSS" field.

Figure 3–2   Download the Current CSS
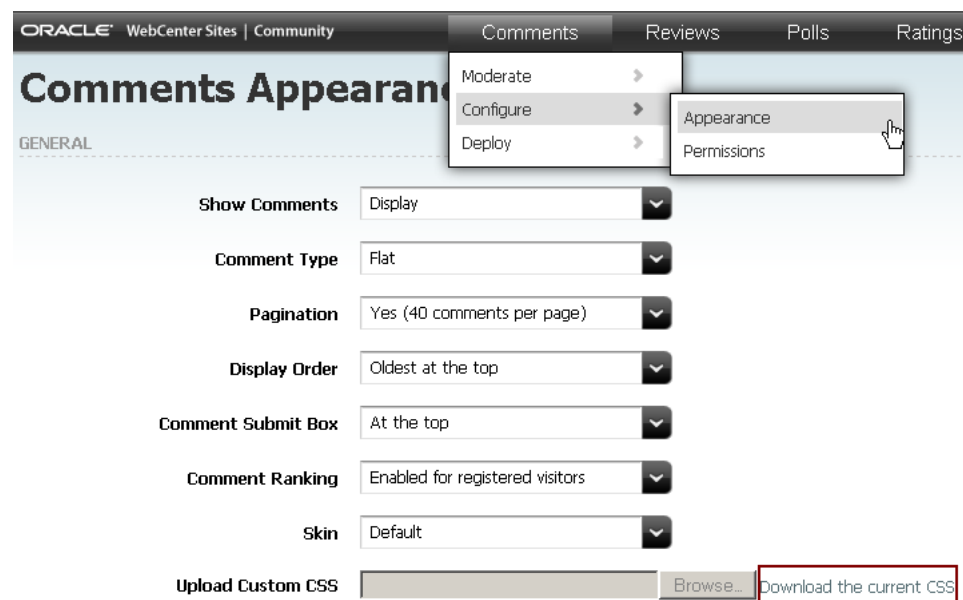
4. Open the downloaded CSS in FireFox for editing.

5. From the "View" menu, choose **Firebug**.

6. From the "FireBug" menu, choose **Inspect Element**. The CSS should look like the CSS in Figure 3–3.

Figure 3–3 CSS Opened in FireBug

7. The DIV element, which contains the entire Comments code, includes the `wsdk-records-record` class. Search for this class in the default CSS previously downloaded by finding the following:

```
.wsdk-records .wsdk-records-record
{
border-top: 1px dashed #666;
clear: both;
overflow: hidden;
margin: 5px 0px;
}
```

Now, let's apply some customizations, for example change the default text color and size, and border:

```
.wsdk-records .wsdk-records-record
{
border-top: 2px solid orange;
color: #359AD3;
font-size:15px;
clear: both;
overflow: hidden;
margin: 10px 0px;
}
```

We can also find that background color is defined in the next div inside this container div, marked with CSS class "wsdk-state-default". Let's also define a custom background color:

```
.wsdk-state-default
{
background: silver;
}
```

8. Customize the `wsdk-records-record` class as required (Figure 3–4), then save your changes.

Figure 3–4   Modified CSS
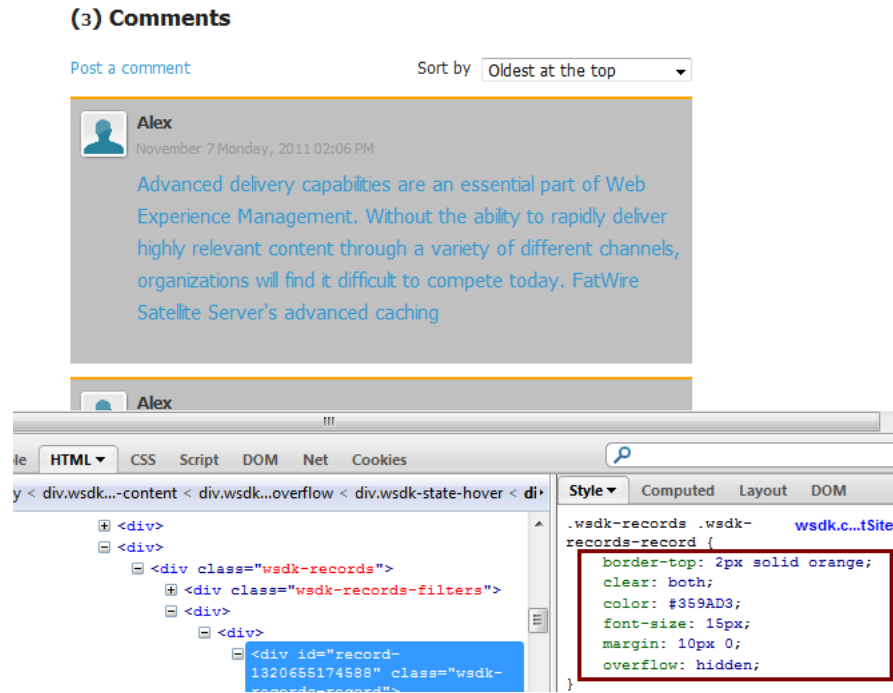
9. To upload the customized CSS, go back to the Community interface.

10. From the "Comments" menu, choose **Configure**, then **Appearance** to display the "Comments Appearance" page.

11. In the "General" section, click **Browse** next to the "Upload Custom CSS" field, and select the CSS file you just customized.

12. Click **Save**.

13. Refresh the web page to refresh the widget, and verify if the changes you made to the CSS reflect in the widget interface.

To customize the CSS for the Reviews widget, go to **Reviews** > **Configure** > **Appearance**, then follow the procedure described above (from step 3 to step 12).

3.2.1.2 Customizing Other Widgets

For customizing widgets other than Comments and Reviews, you can use the general approach described in this section. Depending on your use cases, you can even customize Comments and Reviews widgets using the approach explained here.

The main difference between the approach that will be discussed in this section and what is described for Comments and Reviews widgets is that the customized CSS for other widgets is no longer hosted on the Community application. Typically, the customized CSS is hosted on the web site on which comments are deployed.

Let's customize the Top Ranked Topics widget, which is an add-on to the Reviews functionality.

- Copy the Deployment and CSS Tags Into the Page Template

■ Customize the CSS

Copy the Deployment and CSS Tags Into the Page Template

1. Log in to the Community application as an administrator.

2. From the "Reviews" menu, choose **Deploy**, then **Top Ranked Topics**.

Figure 3–5   Top Ranked Topics Option
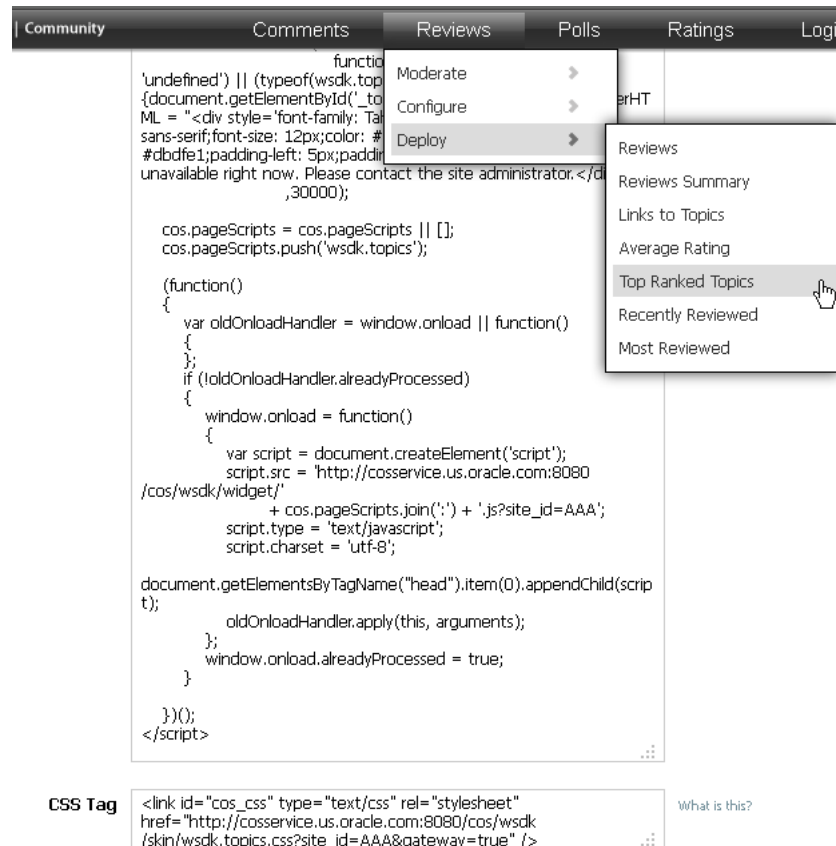
3. Copy and paste into the web page template the contents of both "Tag" and "CSS Tag" fields. Insert the contents of the "CSS Tag" field into the <head> section of page template.

4. Deploy the code snippets. For more information about deploying CSS tags, see the appendix "Deploying the CSS Tag" in the *Oracle WebCenter Sites User's Guide for the Community Application*.

   After the page is rendered, the default look-and-feel of the top ranked topics will be similar to Figure 3–6.

Figure 3–6   Top Ranked Topic After the Tags are Placed in the Template

Customize the CSS

First, take a look at the ID and HREF attributes in the contents of the "CSS Tag" field that you just deployed. The value of the ID attribute must remain unaffected from customization. The HREF value will change as you will learn in the steps described in this section.

```
<link
id="cos_css"
type="text/css"
rel="stylesheet" href="http://localhost:8280/cos/wsdk/skin/wsdk.topics.css?site_
id=FirstSiteII&gateway=true"
/>
```

Before you customize the CSS, download its contents by accessing the HREF location via an Internet browser.

1. In the downloaded CSS, locate CSS classes for topic links. Examine the structure of the widget in FireBug (Figure 3–7). From the "FireBug" menu, choose **Inspect Element**.

Figure 3–7   CSS Opened in FireBug
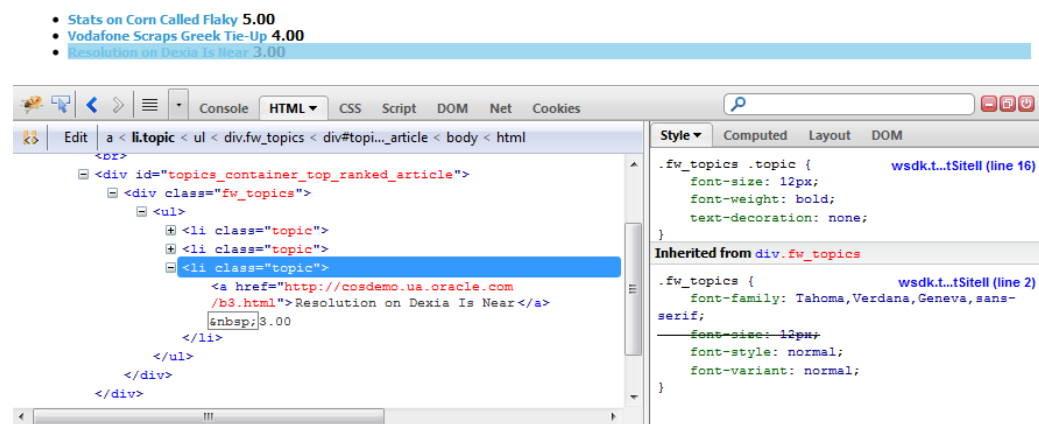
2. To customize the topic headline and topic link colors, locate the respective code snippets. Search for `.fw_topics .headline` and `.fw_topics .topic`.

3. Modify the following code based on web site design specifications:

```
.fw_topics .headline h1 {
color: #359AD3;
font-size: 16px;
}
.fw_topics .topic {
font-size: 14px;
padding: 5px;
font-weight: bold;
text-decoration: none;
}
.fw_topics .topic a {
    color: orange;
    font-size: 12px;
    text-decoration: none;
}
```

4. To apply the CSS to the web site:

    a. Copy it to the folder in which web site styles are stored.

    b. In the CSS you copied to the page template earlier, replace the default location with the new URL from which the customized CSS is accessible. For example, if you copied it to the "skins" folder, then the HREF element should look like this:

```
<link
id="cos_css"
type="text/css"
rel="stylesheet" href="http://localhost:8180/cs/skins/wsdk.topics.css"
/>
```

> **Note:** The ID attribute must remain equal to cos_css because when the Community application loads any widget, it searches on the page for the <link> element with the cos_css ID. If this ID exists, the application continues to render the widget. If this ID does not exist, the application makes an explicit request for the default CSS skin and applies it to the page. Therefore, using cos_css as the ID attribute value prevents the Community application from overwriting the applied customization.

5. Reload the page which uses the widget whose CSS you just modified, and verify that your customizations are applied (Figure 3–8).

Figure 3–8   Customization Applied to a Web Site Page

- **Stats on Corn Called Flaky 5.00**
- **Vodafone Scraps Greek Tie-Up 4.00**
- **Resolution on Dexia Is Near 3.00**

> **Tip:** When deploying Community widgets on the same page, optimize the number of network calls by loading all of the widgets' CSS files in a single network call. The Community application supports this option by allowing you to specify a colon-separated list of widget names in the <link/> tag. That is, widget names do not need to be in separate <link/> elements. For example:
>
> ```
> <link
> id="cos_css"
> type="text/css"
> rel="stylesheet" href=" http://localhost:8280/cos/wsdk/skin/<widget
> name #1>:<widget name #2>:<widget name #3>.css?site_
> id=FirstSiteII&gateway=true "
> />
> ```
>
> Examples of widget names are wsdk.topics and comments-summary. You can find out the actual values for a particular widget in the "CSS Tag" field of the corresponding deployment page.

3.2.2 Customizing a Widget Template

In the case of dramatic changes (such as changing the position of action links from bottom to top in the Comments widget), widget templates can be customized according to project requirements.

Widget templates are rendered dynamically using JavaScript on the browser side. These templates are based on the Google Closure Templates technology.

There is a set of template directives that can be used in customizations. For information, visit the Google documentation web site at:
http://code.google.com/closure/templates/docs/commands.html

Ensure that you have read and understood the `print`, `if/else`, `for`, and `foreach` statements in Google documentation before you apply the information discussed in this section.

This section includes the following:

- Section 3.2.2.1, "Understanding Community Widgets Templates"

- Section 3.2.2.2, "Creating a Sample Template"

- Section 3.2.2.3, "Loading Custom Data Sets"

3.2.2.1 Understanding Community Widgets Templates

This section describes widget template technology and syntax. It explains how you can override attach points by translating those declared in the Community application's templates into corresponding WebCenter Sites template names. This section provides information about attach points available for customization in the Community application, and it explains how to navigate to the widget structure and locate the necessary attach points when customizing a template.

This section includes the following:

- Section 3.2.2.1.1, "Context Variable Access Points"

- Section 3.2.2.1.2, "Dynamic Scripting"

- Section 3.2.2.1.3, "Widget Sources and Templates"

- Section 3.2.2.1.4, "Model-View-Controller Pattern"

- Section 3.2.2.1.5, "Model-View-Controller Regions"

- Section 3.2.2.1.6, "Nested Templates"

- Section 3.2.2.1.7, "Customization Workflow"

- Section 3.2.2.1.8, "Attach Points in the Widget Template Structure"

3.2.2.1.1 Context Variable Access Points To modify templates, developers need to work with visitors' permissions (configured in the Community interface) and variables. Each Community application template includes a standard `$stack` variable that you can use to discover these context variables and permissions. For example, you can use the following tag to insert localizable resources into a template:

```
<h1>Welcome to {$stack.resources.get('label.comments')} Widget </h1>
```

The `$stack.resources` variable is an access point to all the resources declared in `cos.war/WEB-INF/classes/i18n_resources/widgets/cos.commons` and `cos.war/WEB-INF/classes/i18n_resources/widgets/<widget name>`.

3.2.2.1.2   Dynamic Scripting You can build the dynamic logic in the template by using JavaScript's {script} directive:

```
{script}
var lastIndex = 10;
{/script}
```

Then, variables defined inside {script} are available in other closure directives with $$ mark; for example, $$lastIndex:

```
{for $index in range($$lastIndex)}
For loop: Iteration #{$index + 1} of {$$lastIndex} in total
<br />
{/for}
```

In a scenario when the Google closure template variable needs to be accessed in the {script} tag, the regular {$var} syntax can be used:

```
{for $index in range(2)}
   {script}
          alert({$index});
   {/script}
{/for}
```

The data produced inside {script} can be displayed on a web page in the following ways:

- By closing the {script} tag and printing the data previously declared using the {$$variable}} syntax.

- By using the system output variable provided by the Community application to print results immediately from JavaScript using output.append(variable) inside the {script} statement:

```
{script}
        var message= "Hello World!";
        output.append(message);
{/script}
```

3.2.2.1.3   Widget Sources and Templates The list of widget sources shipped with the Community application is located in the cos.war/js/widgets directory. For example, the Comments widget's code is stored in the cos.war/js/widgets/wsdk.comments directory. Each widget directory contains a .shtml file, the widget layout definition, and the main entry point to the process of rendering the widget interface. For example, the comments_layout_view.shtml file is the main entry point for the Comments widget.

3.2.2.1.4   Model-View-Controller Pattern To build a widget interface, the Community application uses the Model-View-Controller (MVC) pattern. Therefore, each interface region includes three items corresponding to Model, View, and Controller. For example:

```
comments_layout_action.js (Controller)
comments_layout_model.js (Model)
comments_layout_view.shtml (View)
```

3.2.2.1.5   Model-View-Controller Regions To build a widget interface, the Community application works with MVC definitions instead of template (.shtml) definitions.

For example, the Comments layout's MVC region can be identified by the `/comments_` `layout` ID in the `cos.war/js/widgets/<WidgetName>/...` directory. The `"/"` special character signifies a relative path in the widget folder. The `comments_layout` MVC prefix is used with suffixes such as `_view.shtml` (`view` contains HTML pages with bindings) for viewing and discovering other corresponding MVC entities. So, the MVC definition of a particular interface region is identified by its relative path in its widget folder and the prefix of MVC entities that are used to discover a particular model, view, or controller.

Understanding such MVC IDs is essential in building widgets interfaces because these IDs are heavily used across templates.

3.2.2.1.6 Nested Templates When an MVC entity is processed and its template is rendered, another MVC entity can be recursively included in the template so that complex interfaces can be built by aggregating smaller interface pieces.

To enable nesting of templates, you can add the following tag to a template in which you want to nest other templates. For example, nesting the Login Bar widget in the Reviews widget.

```
<div attachPoint="/sample"></div>
```

When you include this tag in your template, the Community application searches for the `sample_action.js` and `sample_view.shtml` files in the widget's root folder: `cos.war/js/widgets/<WidgetName>/`.

In addition to the `attachPoint=""` syntax, the `bindMvc=""` directive also includes nested templates, for example `bindMvc="wsdk.ui.input"`. However, this directive calls the templates that are located outside of a particular widget folder hierarchy, and therefore, does not include templates from the current widget folder hierarchy. This enables the reuse of components across multiple widgets in the Community application.

3.2.2.1.7 Customization Workflow The previous section discussed how you can use attach points to nest templates. This section describes how to customize or override attach points via templates.

When the Community application compiles a JavaScript bundle for a widget, it downloads from WebCenter Sites any widget template customizations that are defined as a WebCenter Sites template, compiles them into JavaScript, and applies them to the widget. A contract for template names, which allows to make an association between the Community application and WebCenter Sites, is specified at the following location: `http://{host}:{port}/cs/ContentServer?pagename={site name}/cos/{widget}/{attach point}_view.shtml&ft_ss=true`. It also enables the Community application to discover specific templates.

To customize a template, for example the `/sample` template, you must first create the WebCenter Sites template asset on the production WebCenter Sites system on which the Community application is configured.

To translate an attach point into a WebCenter Sites template name, use the following format:

```
http://{host}:{port}/cs/ContentServer
?pagename={site name}/cos/{widget}/{attach point}_view.shtml
&ft_ss=true
```

For instance, in one of the templates of the "hello-world" widget, the `attachPoint="/sample"` attach point is translated into the following HTTP request to WebCenter Sites:

```
http://{host}:{port}/cs/ContentServer
?pagename={site name}/cos/hello-world/sample_view.shtml
&ft_ss=true
```

For a custom Community application template, you must create a corresponding template in WebCenter Sites by following the WebCenter Sites naming convention: "`cos/hello-world/sample_view.shtml`".

The `bindMvc` is a set of pre-defined attach points that can be potentially used for nesting templates. Table 3–3 lists the templates that you can nest using the `bindMvc=""` attribute syntax in the template you just created. You can find the default templates in the corresponding subfolders in the `cos.war/js/widgets.commons` directory in which reusable widget code is stored.

To override a component, use the mapping (Table 3–13) for binding the MVC names to the names of templates that you will create in WebCenter Sites.

Table 3–13   Mapping for Binding MVC Names To Template Names

Bound MVC	Template Name to Override
`wsdk.ui.stars`	`cos/wsdk.ui.stars/stars_view.shtml`
`wsdk.ui.thumbs`	`cos/wsdk.ui.thumbs/thumbs_view.shtml`
`wsdk.ui.input`	`cos/wsdk.ui.input/input_view.shtml`
`wsdk.ui.textarea`	`cos/wsdk.ui.textarea/textarea_view.shtml`
`wsdk.ui.pagination`	`cos/wsdk.ui.pagination/pagination_view.shtml`
`wsdk.ui.session`	`cos/wsdk.ui.session/session_view.shtml`

3.2.2.1.8   Attach Points in the Widget Template Structure The interfaces and functionality of Community widgets are built with the help of templates structured in a certain way. Figure 3–9 shows the structure of the Comments widget. You can treat this structure as a navigation map while you are working with customizations.

Figure 3–9   Structure of the Comments Widget

The attach points that are invoked and rendered dynamically by following a visitor's action are not referenced directly in templates. For example, the attach points that are invoked at the time when a visitor clicks "edit" or "delete" links on a comment. The dialog boxes for these events cannot be rendered during the initial widget load phase, and therefore, they are loaded programmatically later.

Some examples of attach points for such dynamic dialog boxes are:

■   The "Delete" dialog box (Figure 3–10) is displayed when a visitor clicks the "delete" link on a comment.

Figure 3–10   Delete Dialog Box

- The "Flag Content" dialog box (Figure 3–11) is displayed when a comment is flagged.

Figure 3–11   Flag Content Dialog Box

For the Reviews widget, the templates and their attach points are structured as shown in Figure 3–12. The attach points for flagging, editing, and removing a review are the same as for comments: "/record_list/record/flag/flag", "/record_list/record/edit/form", and "/record_list/record/remove/remove", respectively.

Figure 3–12   Post a Review Page

Forms for posting comments and reviews contain the "Preview" button. Clicking this button displays a pop-up with the rendered comment or preview (Figure 3–13).

Figure 3–13   Preview Pop-Up Dialog Box

For both, comments and reviews, the "/record_list/record/preview/preview" attach point creates this pop-up functionality.

> **Tip:** The Community application loads customized templates from the production WebCenter Sites instance. However, the production instance may not have editorial interface to manage these templates. Therefore, it is recommended to start with the development environment first, and point production and management Community application server instances to the single instance of WebCenter Sites that has the editorial interface. Using this approach, you can compose templates and verify their look-and-feel in the Community application. Once you have created and tested your templates, you can either export/import them from development environment to production or publish your custom templates to the desired location.

3.2.2.2 Creating a Sample Template

To create a sample template using the WebCenter Sites Advanced interface:

1. Stop all Community application servers, including those on production and management environments.

2. Start the production WebCenter Sites instance.

   In the following steps, it is assumed that this server is running on the local host at port 8080.

3. Log in to WebCenter Sites at http://localhost:8080/cs/login, then select the site in which templates need to be customized.

4. Launch the Advanced interface application on that site.

5. To create a customizable template, click **New**.

Figure 3–14   WebCenter Sites Advanced Interface - New

6. In the table on the right side, click the **New Template** link next to "Template".

Figure 3–15   New Template Link

7. Set "Assignee" to any value as this value is not relevant here, then click **Continue**.

8. In the "Name" field, specify the attach point to be overridden in the form `cos/{widget}/{attach point}_view.shtml`.

9. In the "For Asset Type" field, choose **can apply to various asset types**, then click **Continue**.

10. In the "Usage" field, choose **Element defines a whole HTML page and can be called externally**.

11. In the "Create Template Element?" field, click **JSP**.

12. In the "Element Logic" field, just before the `</cs:ftcs>` closing tag, enter the content of the template. The template content must have the following statement, as well as its essentials for invalidating templates cached in the Community application:

```
<%-- 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>
```

13. For the content of the template, navigate to the Community application file system and copy the default content available at `cos.war/js/widgets/{widget}/{attach point}_view.shtml`.

14. Click **Continue**.

15. On the "Site Entry" page, click **Save**.

16. To verify that the newly created template is available to the Community application over HTTP, enter the template's URL in a browser in the following format:

```
http://{host}:{port}/cs/ContentServer?pagename={site}/cos{attach point}_
view.shtml&ft_ss=true
```

17. Start Community application servers, and verify that the customization has been applied to the deployed widgets.

3.2.2.3 Loading Custom Data Sets

In some scenarios, it is necessary to load the customized project-specific data in the existing templates. You can achieve this by exposing the data to the Community application as a REST service, in the JSONP format.

Once you have created a REST service, invoke it from the template using the system loader variable. To ensure that the custom data is loaded before the template is rendered, you must include the invocation in the {script.preload} section at the beginning of the page.

```
{script.preload}
        loader.loadData(url, params, key);
{/script.preload}
```

- `url` - URL address of the JSONP endpoint that provides data.

- `params` - parameters to be passed as GET parameters to the JSONP endpoint, if the query needs to be parameterized.

- `key` - result set returned with data are included in this variable so that this variable is accessible at `$stack.key`.

1. As an example, create a `data.jsp` file which you will use as the JSONP REST service endpoint. Include the following contents in this file:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%
out.write(request.getParameter("cosrestcallback"));
%>({ responseCode : "OK", data : ["foo","bar"]});
```

In this example, the response is provided in the JSON format, containing sample data as a response code and the data array. The JSON response is wrapped up by the callback name according to the JSONP protocol used by the Community application.

2. Deploy the `data.jsp` file to the Community application's `webapp` folder.

3. The Community application has a sample widget that you can use as a sandbox for experimenting with the widget technology. Its source is located in the `cos.war/js/widget/sample.widget` code. To create custom template in WebCenter Sites with the `cos/sample.widget/layout_view.shtml` name, follow the procedure described in Section 3.2.2.2, "Creating a Sample Template."

4. In the customized template, download the data given in the `script.preload` section and then print it on a page.

5. Replace the Community application URL with your location.

Figure 3–1 shows the code for sample data after it is loaded.

Example 3–1   Sample Data Loaded

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"
%><%@ taglib prefix="asset" uri="futuretense_cs/asset.tld"
%><%@ taglib prefix="assetset" uri="futuretense_cs/assetset.tld"
%><%@ taglib prefix="commercecontext" uri="futuretense_cs/commercecontext.tld"
%><%@ taglib prefix="ics" uri="futuretense_cs/ics.tld"
%><%@ taglib prefix="listobject" uri="futuretense_cs/listobject.tld"
%><%@ taglib prefix="render" uri="futuretense_cs/render.tld"
%><%@ taglib prefix="siteplan" uri="futuretense_cs/siteplan.tld"
%><%@ taglib prefix="searchstate" uri="futuretense_cs/searchstate.tld"
%><%@ page import="COM.FutureTense.Interfaces.*,
                   COM.FutureTense.Util.ftMessage,
                   COM.FutureTense.Util.ftErrors"
%><cs:ftcs><%-- /cos/sample.widget/layout_view.shtml
--%>
<%-- 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>
        {script.preload}
             loader.loadData("http://localhost:8280/cos/data.jsp",{},
"customData");
        {/script.preload}
        <div>
<h1>Welcome to Sample Widget. Data Load Demo</h1>
<br />
Data loaded with response code = "{$stack.customData.responseCode}"
<br />
Custom data loaded:  
{script}
    var data = {$stack.customData.data};
    for(var dataIdx in data)
    {
    var item = data[dataIdx];
    {/script}
    {$$item},   
    {script}
    }
{/script}
</cs:ftcs>
```

3.3 Creating a Custom Word Filter

The Community application enables visitors to post their feedback on a web site on which the Community widgets are deployed. To prevent publishing of objectionable content, spam, or any fraudulent information on the web site, the Community application provides a User-Generated Content (UGC) filtering sub-system for content filtration.

The default configuration of the filtering system relies on a word filter file – a list of prohibited words – that can be uploaded via the Community interface > **Settings** > **Restricted Words**. Once the list of prohibited words is uploaded and auto-moderation is enabled via the Community interface > **Settings** > **Moderation**, the Community application examines the content posted by visitors and compares each word with those in the list. If a posted comment or review contains any of the prohibited words, then it is marked "Inappropriate" and is not published on the web site.

> **Note:** To ensure that prohibited words containing special characters are marked "Inappropriate" and not "Approved", you must create a custom word filter with the syntax shown in Example 3–2.

There may be a need to customize the behavior of the default content filter. For example, to integrate with third-party libraries that provide a functionality for statistical inferences and advanced techniques for spam detection and prevention, the Community application provides a pluggable word filter model. It allows you to create a custom word filter from scratch.

To create a custom word filter, you must first create a Java class containing the word filter logic, compile this class, and then assemble it as a JAR file so it can be plugged into the Community application deployment.

1. Create a new Java project in your IDE.

2. Add a JAR library, which contains the `com.fatwire.cos.moderation.filter.WordFilter` interface, to your project's classpath. You can copy this library JAR from the Community application's web application at `cos.war/WEB-INF/lib/cos-api-1.5.jar`.

    Example 3–2 provides the WordFilter interface.

Example 3–2   New Java Project

```
package com.fatwire.cos.moderation.filter;
import java.util.Set;
/**
 * The word filter interface that needs to be implemented
 * when developing and plugging-in a custom word filter
 * to Community Server.
 * The content is assigned to "Inappropriate" category and isn't
 * shown on web site once any of the filters registered in the system
 * reports an abuse.
 * @author e_shevchenko
 *
 * May 16, 2011
 */
public interface WordFilter
{
    /**
     * The filter method makes a decision whether content
     * specified can be shown on web site or not
     * @param text the content submitted by visitor
     * @param profanityWords the list of prohibited words uploaded on
     * "Moderation" page in the Community interface
     * @return true if an abuse detected and false if content
     * can be shown on web site
     */
    boolean filter(StringBuffer text, Set<String> profanityWords);
}
```

3. Create a new Java class in your project, for example, `cos.demo.DemoFilter`.

4. Declare that this class implements the WordFilter interface:

    ```
    package cos.demo;

    import java.util.Set;
    import com.fatwire.cos.moderation.filter.WordFilter;
    ```

```
public class DemoFilter
     implements WordFilter
{
     @Override
    public boolean filter(StringBuffer text,
                          Set<String> profanityWords)
   {
      // TODO add filter logic here
      return false;
   }
}
```

5. Add your custom implementation inside the `filter` method body.

   The `filter` method takes two parameters: the source text submitted by visitors and the list of prohibited words uploaded as a text file via the Community interface. Based on your requirements, you can decide whether to use the existing list of prohibited words or use a third-party database of prohibited words.

   The following example shows how to verify whether user-generated content contains the word "demo":

```
package cos.demo;
import java.util.Set;
import com.fatwire.cos.moderation.filter.WordFilter;
public class DemoFilter
    implements WordFilter
{
    @Override
    public boolean filter(StringBuffer text, Set<String> profanityWords)
    {
        boolean result = false;
        if(null != text )
        {
            result = text.toString().contains("demo");
        }
        return result;
    }
}
```

6. Implement and then compile the contents of the filter class.

7. Package the compiled class as a JAR file, `filter-demo.jar`.

8. To make this custom filter available to the Community application's class loader, copy the `filter-demo.jar` file to the `cos.war/WEB-INF/lib` directory in both, management and production environments.

9. To enable the Community application to discover the custom filter, plug this filter into the `cos_word_filters.xml` configuration file located in the `cos.war/WEB-INF/classes` directory:

```
<?xml version="1.0" encoding="UTF-8" ?>
<word-filters>
<word-filter
className="com.fatwire.cos.records.moderation.filter.DefaultWordFilter"
/>
</word-filters>
```

10. Replace the default word filter. On both management and production environments, remove the existing `<word-filter/>` element from the `cos_word_`

filters.xml files and add a new one that contains a reference to the class you created. The contents of the new cos_word_filters.xml files will look like the following:

```
<?xml version="1.0" encoding="UTF-8" ?>
<word-filters>
    <word-filter className="cos.demo.DemoFilter"/>
</word-filters>
```

11. Restart the Community application servers on both, management and production sytems.

12. To enable automatic word filtering, go to the Community application's interface > **Settings** >**Moderation** and select **Auto-moderate against Restricted Words** in the "AUTO-MODERATION "section (Figure 3–16).

Figure 3–16   Auto Moderation Against Restricted Words
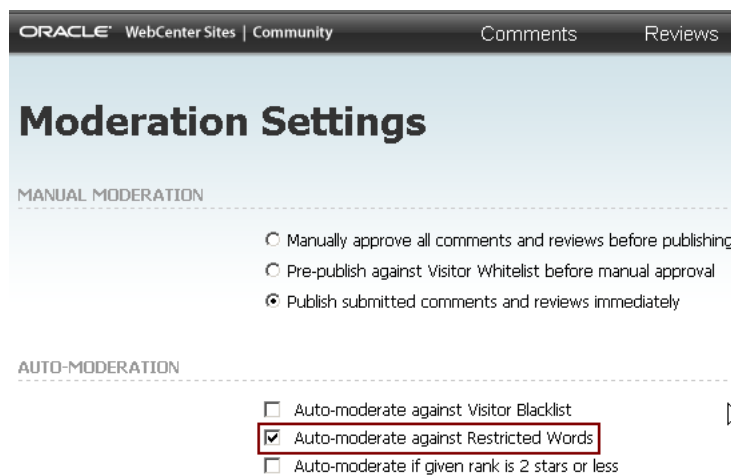
13. On the web site, post a comment or review which includes the word "demo". You will notice that your post will be assigned to the "Inappropriate" category, and therefore, it will not be published.

3.4 Creating a CAPTCHA Generator

CAPTCHA technology helps identify robots of automated spamming systems that may attack a web site. The Community application provides a plug-in model for custom implementations of CAPTCHA.

This section describes the procedure for creating a simple CAPTCHA generator and plugging it into the Community application. This procedure is similar to the procedure for creating a custom word filter. To create a CAPTCHA generator, you will create a new class which implements the required interface, compile this class, package it as a JAR file, and then deploy it to Community application server.

1. Create a new Java project, "demo-captcha" in your IDE.

2. Add a JAR library for your project to your project's classpath. You can copy the required JAR from the Community web application directory available here: cos.war/WEB-INF/lib/cos-api-1.5.jar.

3. From the `cos-api-1.5.jar` file, extract the interface `com.fatwire.cos.captcha.CaptchaGenerator` to be implemented. Example 3–3 shows the contents of this interface.

Example 3–1   com.fatwire.cos.captcha.CaptchaGenerator Interface

```
package com.fatwire.cos.captcha;
/**
 * The interface that needs to be implemented when
 * creating plug-in that generates CAPTCHA to be shown to
 * visitors when UGC is submitted.
 *
 * It's a factory that generates complex Captcha objects that are managed
 * by Community Server and utilized when CAPTCHA is rendered for visitors
 * and when it's validated on content submission
 *
 * @author alex
 *
 * Oct 17, 2011
 */
public interface CaptchaGenerator
{
    /**
     * Factory method that creates a new captcha object that consists of challenge
text
     * and challenge image to shown to visitors
     * @return
     */
    public Captcha generate();
}
```

4. In the "demo-captcha" project, create a new Java class called `cos.demo.DemoGenerator` (Example 3–4).

Example 3–4   demo-captcha Project

```
package cos.demo;
import com.fatwire.cos.captcha.Captcha;
import com.fatwire.cos.captcha.CaptchaGenerator;
public class DemoGenerator
    implements CaptchaGenerator
{
    @Override
    public Captcha generate()
    {
        // TODO add implementation
        return null;
    }
}
```

5. Implement the `CaptchaGenerator` interface.

6. Implement the `generate` method, as shown in Example 3–5.

   First, generate the challenge text, and then create the image displaying the generated text. Once text and image are ready, package them into the Captcha object with the help of the `CaptchaFactory` class. It is highly recommended that you use this factory instead of creating a custom implementation of the Captcha object. This is because the Captcha object built by the factory is already serializable

and can be safely shared across cluster members. If you choose to create a custom implementation, then make the Captcha object serializable.

Example 3–5   Generate Method

```
package cos.demo;
import com.fatwire.cos.captcha.Captcha;
import com.fatwire.cos.captcha.CaptchaFactory;
import com.fatwire.cos.captcha.CaptchaGenerator;
public class DemoGenerator
    implements CaptchaGenerator
{
    @Override
    public Captcha generate()
    {
        String challenge = "foo_bar";// TODO add generation
        byte[] image = generateImage(challenge); //TODO: generate image
        return CaptchaFactory.create(challenge, image);
    }
}
```

7. Package the compiled class into a JAR file, `captcha-demo.jar`.

8. To make the generator available to the Community application class loader, copy the `captcha-demo.jar` file to the `cos.war/WEB-INF/lib` directory on both management and production environments.

9. To enable the Community application to discover the generator, plug it into the configuration file. Create the `cos_captcha.xml` file in the `cos.war/WEB-INF/classes` directory with the following contents:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<captcha className="cos.demo.DemoGenerator"/>
```

10. Restart the Community application servers on both, management and production systems.

11. To enable CAPTCHA for comments, go to the Community interface, select **Comments** > **Configure** > **Permissions**, then select **User must enter a verification code** in the "Who Can Comment?" section (Figure 3–17).

Figure 3–17   Enable CAPTCHA

12. Repeat the previous step to enable CAPTCHA for reviews.

13. Deploy Comments or Reviews widgets with login bar support.

14. To verify that the field for the CAPTCHA challenge is displayed and the CAPTCHA image is rendered, click the **Register** link on the web site. You can also try to post a comment or review as a guest. CAPTCHA will be displayed in the form.

4

Securing the Community Application

This chapter describes how to enable security between the Community application and visitor CAS application so transactions made via third-party authentication providers are secured. This chapter includes the following sections:

4.1 About Security

Third-party authentication providers such as Facebook and Twitter rely on the security features of the application through which the third-party authentication support is enabled. These authentication providers let the application security validate the user name and password of a visitor when a user session is starting. To prevent an intruder from intercepting requests between the Community application and visitor CAS, a secured bridge can be established with the help of assymetric cryptography. The Community application comes packaged with default pre-generated RSA certificates. While these default certifications can be used for demo purposes, the development and QA environments must generate new certificates for production deployments.

> **Note:** Because RSA certificates are used to secure a communication channel, it is very easy to mistake them for SSL certificates and for the configuration which is typically performed when configuring products to work over HTTPS. Do not mix these activities as they serve different purposes.

Both, the Community application and the visitor CAS application have the following components:

- A key storage (Java Key Store, JKS) each, which is packaged with its own private and public key pair.

- The public key (certificate) of the application to enable communication between both Community applications.

The Community application contains its own JKS, `cos.jks` file and the visitor CAS application's public key, the `cas.crt` file. Similarly, the visitor CAS application

contains its own JKS, the `cas.jks` file and the Community application's public key, the `cos.crt` file. These files are located in the `WEB-INF/classes` directories of both applications.

Access to the JKS file that contains the private key is protected by a password. We recommend changing the default password for security reasons.

4.2 Generating Security Certificates

This section describes how to generate new certificates to replace default certificates.

The `keytool` Java utility, which is located in the *JAVA_HOME*/bin directory, is used to generate new certificates. This is a command line utility, and you need to supply a number of arguments when running it. The notation is as follows:

```
keytool -genkeypair -keystore keystore -storepass storepass -alias alias -keyalg
keyalg -dname dname -keypass keypass -validity valDays
```

This command generates a key pair (a public key and associated private key). It wraps the public key into an `X.509 v3` self-signed certificate, which is stored as a single-element certificate chain. This certificate chain and the private key are stored in a new `keystore` entry identified by *alias*.

Please see the official documentation for more details: http://docs.oracle.com/javase/6/docs/technotes/tools/windows/keytool.html

While the `keytool` utility requires many arguments, the following are mandatory for the Community application:

- `keystore keystore`: The target keystore file that will be generated.

- `storepass storepass`: The password to protect the integrity of the keystore. The `storepass` password must be at least six characters long. This password is required in all subsequent operations performed on keystore contents. If you do not set the `-storepass` option at the time when you generate key certificates, you will be required to supply it when prompted for it.

- `keyalg`: The algorithm for generating the key pair. For the Community application, `RSA` must be used.

- `dname dname`: The `X.500` *distinguished name* to be associated with alias. It is supplied to the issuer and subject fields in the self-signed certificate. If you do not provide a distinguished name at the time when you generate new certificates, you will be prompted to specify it.

- `keypass keypass`: The password used to protect the private key of the generated key pair. If you do not provide a password when you generate new certificates, you will be prompted to specify it.

  > **Note:** Pressing the **RETURN** key at the prompt sets the same password as that used for the keystore.
  >
  > `keypass` must be at least six characters long.

- `validity valDays`: The number of days for which the certificate should be considered valid.

Sample commands may look like the following:

For the Community application:

```
keytool -genkeypair -keystore cos.jks -storepass cosstorepass -alias coskey
-keyalg RSA -dname "CN=cos, O=FatWire, C=US" -keypass coskeypass -validity 360
```

For the visitor CAS application:

```
keytool -genkeypair -keystore cas.jks -storepass casstorepass -alias caskey
-keyalg RSA -dname "CN=cas, O=FatWire, C=US" -keypass caskeypass -validity 360
```

4.3 Exporting Certificates From JKS Files

After generating JKS files for the Community application and the visitor CAS application, extract the public key certificate from each file. The Community application's certificate is shared with the visitor CAS application, and vice versa.

To export the certificate from keystore, run the following command on the Community application and the visitor CAS application using values appropriate to each of these applications:

```
keytool -exportcert  -keystore keystore.jks -storepass storepass  -alias alias
-file cert_file.crt
```

This command reads from the `keystore.jks` file the certificate associated by *alias* and saves it into the `cert_file.crt` file. When generating keystores, the `-keystore` and `-storepass` arguments must be the same as those used in previous commands.

Continuing the examples from the previous section, the commands may look like the following:

For the Community application:

```
keytool -exportcert -keystore cos.jks -storepass cosstorepass -alias coskey -file
cos.crt
```

For the visitor CAS application:

```
keytool -exportcert -keystore cas.jks -storepass casstorepass -alias caskey -file
cas.crt
```

4.4 Deploying Certificates to the Community Applications

Once you have generated keystores and exported public certificates to the `cert_file.crt` file, replace default certificates with the newly generated certificates on the Community application and the visitor CAS application.

- The Community application: To replace default certificates with new, upload `cos.jks` and `cas.crt` files to the `cos.war/WEB-INF/classes` directory.

- The visitor CAS application: To replace default certificates with new, upload `cas.jks` and `cos.crt` files to the `cas.war/WEB-INF/classes` directory.

4.5 Configuring the Community Application

Once you have deployed the certificates, the final step is to specify the keystore and the certificate password along with the certificate *aliases* in the applications' configuration. On the Community application and the visitor CAS application, update

the configuration in the setup_security.properties file located in the
WEB-INF/classes directories of cos.war and cas.war applications.

> **Tip:** As these files are identical in both applications, you can update
> the setup_security.properties file for one application and then
> copy it to another application's WEB-INF/classes directory.

```
################################################################################
#################
#### Section 1: PRIVATE KEYS SETTINGS
################################################################################
#################
## CoS Private Key Settings
######################################
…
#
# CoS keys storage password. Should be in the encrypted form.
#
widgets.security.cos.attrs.keystore_password=<specify your keystore password for
cos.jks>
#
# CoS private key identity (alias) in the CoS keys storage (Java Key Store)
#
widgets.security.cos.attrs.privatekey_alias=<specify CoS alias, e.g. "coskey">
#
# CoS private key password. Should be in the encrypted form.
#
widgets.security.cos.attrs.privatekey_password=<specify a cos key password>
## CAS Private Key Settings
######################################
…
#
# CAS keys storage password. Should be in the encrypted form
#
widgets.security.cas.attrs.keystore_password=<specify keystore password for
cas.jks>
#
# CAS private key identity (alias) in the CAS keys storage (Java Key Store)
#
widgets.security.cas.attrs.privatekey_alias=<specify CAS alias>
#
# CAS private key password. Should be in the encrypted form
#
widgets.security.cas.attrs.privatekey_password=<specify a cas key password>
```

After updating the setup_security.properties file for both applications, restart the
application servers hosting cos.war and cas.war. Ensure that no security exceptions
occur in the logs of the Community application and the visitor CAS application.

5

Translating the Community Application's Functionality into Different Languages

This chapter describes how to enable translation of the Community application functionality into multiple languages.

This chapter includes the following sections:

- Section 5.1, "About Localization"
- Section 5.2, "Adding a New Language to the Community Application"
- Section 5.3, "Registering the New Language in the Community Application"

5.1 About Localization

The Community application's user interface can be translated, by default, into the following languages: English, French, German, Italian, Japanese, Korean, Portuguese, Spanish, Simplified Chinese, and Traditional Chinese. Moreover, you can also customize existing labels and translations to new languages on demand.

The language pack is located in the file system of the Community application at `cos.war/WEB-INF/classes/i18n_resources`.

Figure 5–1   Language Pack
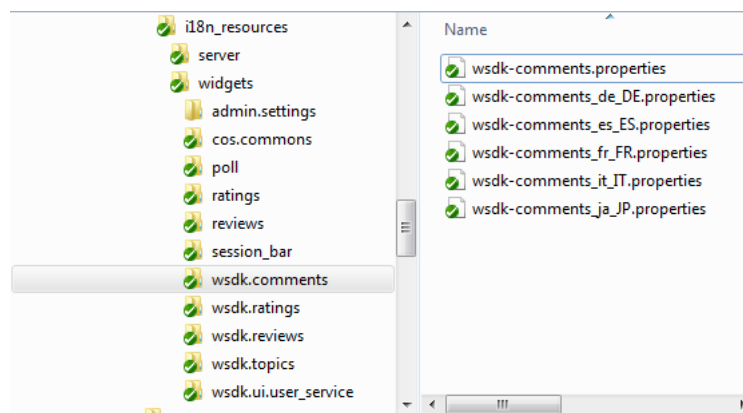
The `i18n_resources` directory contains two subfolders, `server` and `widgets` that include resources for the Community interface and for client-side widgets respectively.

The `server` directory contains a list of Java property resource bundles that are separated on the basis of the Community application sub-system. It contains

localizations for the following sub-systems: `core`, `cos-resources`, `service`, `shared`, and `users`.

The `widgets` directory contains a list of folders in which each folder corresponds to a particular widget. For example, the `wsdk.comments` folder corresponds to the Comments widget and the `wsdk.reviews` folder to the Reviews widget.

The `cos.commons` directory contains resources that are shared across all the widgets.

The logic of picking up a required language is different in the Community application's interface and in the widgets deployed on a web site. The following sections discuss both logics in the order of priority.

Section 5.1.1, "Language Detection for the Community Interface"

- Section 5.1.2, "Language Detection for Community Widgets"

5.1.1 Language Detection for the Community Interface

1. If a language is customized in the Web Experience Management (WEM) profile for the logged-in business user, apply it. If not, then proceed to step 2.

2. If the locale of the browser is supported, apply this setting. If not, proceed to step 3.

3. Apply the default `English` locale.

5.1.2 Language Detection for Community Widgets

Language detection logic described in this section applies to all Community widgets.

The following first and second steps are based on the premise that the web site may already have some language selection mechanism to display site content in different languages. For example, most sites have a language icon on the top or bottom of a site page for this purpose. If this mechanism uses cookies or JavaScript variables for language selection, then the Community widgets can also use them after client's consent.

Site developers customize the language selection mechanism by enabling the creation of the `cos_language` cookie or the JavaScript variable with a language value. Once this is done, the Community widgets can use the client's language selection mechanism to display site content in different languages.

1. Use the language parameter given in a `cos_language` cookie that site developers specify manually.

   If no cookie is set, proceed to step 2.

2. Use the language parameter of a global JavaScript variable, `cos_language` that site developers or integrators can specify manually.

   If no variable is set, proceed to step 3.

3. Use the default language setting chosen by the administrator via the Community interface > **Settings** > **Language** page (Figure 5–2).

   This setting applies to all types of widgets deployed on the site.

Figure 5–2   Language Setting in the Community Application

5.2 Adding a New Language to the Community Application

To add a new language:

1. List the new language in the Community application's configuration (see "Registering the New Language in the Community Application"), so that it displays in the default language selector in the Community interface.

2. Upload translations for all the resources located in `cos.war/WEB-INF/classes/i18n_resources/<subfolders containing translation files>`. That is, translate all `*_en.properties` files located in this directory and copy back the translated files (for example, for Russian language the file is `*_ru.properties`).

5.3 Registering the New Language in the Community Application

To register a new language in the Community application configuration:

1. Navigate to the `cos.war/WEB-INF/lib` directory.

2. Open the `cos-shared-11.1.1.6.0.jar` file, then extract the `cos_core_metadata.xml` file.

3. Search for the `schema::cos::commons:permissions:language` string. The declaration is as follows:

```
<bean id="language"
class="com.fatwire.cos.metadata.core.model.MetadataDescriptorImpl">
      <property name="uid"
                       value="schema::cos::commons:permissions:language"/>
      <property name="name" value="label.language"/>
      <property name="defaultValue" value="en_US"/>
       <property name="dataOptionsValue" value="[
{ name: 'label.language.english', value:'en_US'},
{ name: 'label.language.brazilian_portuguese', value:'pt_BR'},
{ name: 'label.language.chinese_simplified', value:'zh_CN'},
{ name: 'label.language.chinese_traditional', value:'zh_TW'},
{ name: 'label.language.french', value:'fr'},
{ name: 'label.language.german', value:'de'},
{ name: 'label.language.italian', value:'it'},
{ name: 'label.language.japanese', value:'ja'},
{ name: 'label.language.korean', value:'ko'},
{ name: 'label.language.spanish', value:'es'},
<ADD COMMA AND INSERT NEW LANGUAGE HERE>
]"/>
                                            …
```

```
</bean>
```

4. To the `dataOptionsValue` property, add a new JSON object for the new language. For example, for Russian, the entry can be the following:

```
{ name: 'label.language.russian', value:'ru'}
```

If the entry for Russian is added, the property value will be:

```
<property name="dataOptionsValue" value="[
{ name: 'label.language.english', value:'en_US'},
{ name: 'label.language.brazilian_portuguese', value:'pt_BR'},
{ name: 'label.language.chinese_simplified', value:'zh_CN'},
{ name: 'label.language.chinese_traditional', value:'zh_TW'},
{ name: 'label.language.french', value:'fr'},
{ name: 'label.language.german', value:'de'},
{ name: 'label.language.italian', value:'it'},
{ name: 'label.language.japanese', value:'ja'},
{ name: 'label.language.korean', value:'ko'},
{ name: 'label.language.spanish', value:'es'}
{ name: 'label.language.russian', value:'ru_RU'} ]"/>
```

5. Save the `cos_core_metadata.xml` file and include this revised file in the `cos-shared-11.1.1.6.0.jar` file.

6. Navigate to the `i18n_resources/server/cos-resources_<LANG>.properties` resource bundle, then add the `label.langauge.russian` key with value translation for each supported language.

   Each of the `cos-resources` files (`cos-resources.properties`, `cos-resources_de.properties`, `cos-resources_es.properties`, `cos-resources_fr.properties`, `cos-resources_it.properties`, and so on) contains a set of language labels. The following is an example for `cos-resources_en.properties`:

```
label.language.english = English
label.language.brazilian_portuguese = Brazilian Portuguese
label.language.chinese_simplified = Simplified Chinese
label.language.chinese_traditional = Traditional Chinese
label.language.french = French
label.language.german = German
label.language.italian = Italian
label.language.japanese = Japanese
label.language.korean = Korean
label.language.spanish = Spanish
```

   Add a new line containing the parameter name (e.g. `label.langauge.russian`) and the parameter value (e.g. `"Russian"` - for cos-resources_en.properties):

```
label.langauge.russian=Russian
```

   The `cos-resources_en.properties` value for `label.langauge.russian` is different (`"Russische"` in `cos-resources_de.properties` or `"Russe"` in `cos-resources_fr.properties`).

   Therefore,

```
label.langauge.<lang_id>=<value_translation>
```

7. For all resource bundles that can be found recursively in the `i18n_resources` directory, upload translation files in the same folder with the corresponding language suffix. For example, in the `i18n_resources/server/users.properties` directory you can find Russian translation for English pack. So, upload the

translation file for `Russian: users_ru_RU.properties` in the `i18n_resources/server/` directory.

8. Restart the application server for the Community management application.

9. To verify that the new language is displayed, log in to the Community application as an administrator, then choose **Language** from the "Settings" menu.

6

Monitoring Community Application Performance

This chapter provides an overview of caching in WebCenter Sites. It explains how content is fetched and displayed on a web site when cache is enabled or disabled. This chapter also explains different cache modes and how they affect the performance and data consistency of the Community application. It describes how to configure cache and optimize the handling of user-generated content.

This chapter includes the following sections:

- Section 6.1, "About Caching"
- Section 6.2, "Configuring Cache in the Community Application"
- Section 6.3, "Optimizing User-Generated Content (UGC) in the Community Application"

6.1 About Caching

When a system receives a request, it routes it to different sub-systems that perform the required functions to serve this request, as shown in Figure 6–1 and described later in this section. The results of the request are compiled and then served to the client that made the request. These results could be in the form of complete comments for a page, comments sorted in a certain way, reviews, and so on. To improve the application's performance, these results can be stored in a such way that when the system receives a similar request, it can immediately locate and reuse the stored information to serve the new request.

> **Tip:** The Community application uses the WebCenter Sites data repository and communicates with it over REST. WebCenter Sites, in turn, translates the REST request into a database query.

When the Community application loads content from its database, it caches that content in the in-memory cache, which is built with the help of the inCache technology provided by WebCenter Sites. Cache is an intermediate storage (RAM) in which data for the most popular and frequent queries is stored. The access to cache is very fast as RAM is used as storage.

Figure 6–1   Cache Process Flow
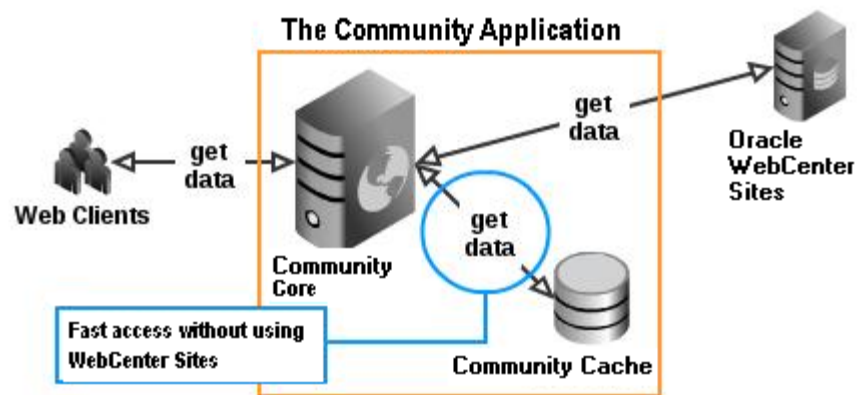
Caching helps save network communication and data serialization overheads. To improve system performance and throughput, requests should be served from cache.

Two types of events take place when cache technology is used to serve requests: cache hit and cache miss. A *cache hit* occurs when the requested data already exists in the cache, and therefore, it is immediately served from the cache to a client. A *cache miss* occurs when the requested data does not exist in the cache. In this case, first the data from the WebCenter Sites data repository is copied into the cache, and then subsequent requests are served. For example, if a user requests the page on which comments/reviews were posted for the first time (that is, this page was not requested before, or cache was flushed), the Community application saves it to the cache. When another user requests this page, the Community application retrieves it from the cache. As a result of interaction between users and the Community application, the application caches all the data loaded from WebCenter Sites. This cache is an in-memory cache.

The following section explains how a system behaves with caching.

6.1.1 The Community Application With Cache

When a visitor creates, updates, or deletes an item such as a comment or review on a web site page, the Community application:

■   modifies all the related data in WebCenter Sites

■   invalidates all the related cache entries

This way, the Community application indicates to the whole system that cached entries are stale or no longer valid, and therefore, they should be updated based on the user's action. For example, when you edit an existing comment, this particular item and the entire data collection associated with the discussion whose comment you edited needs to be invalidated in the cache. The update of invalidated values happens in the background for the future access.

WebCenter Sites supports remote server in which cached content is kept up-to-date by sending invalidation requests over the REST protocol. To enable this, register the Community application as a remote server in the SystemSatellite table. For information, see the chapter "Installing WebCenter Sites: Community" in the *Oracle WebCenter Sites Installation Guide for the Community Application*.

Once you have registered the Community application as a satellite server, it starts receiving notifications about the modified data and invalidates its cache accordingly.

When the data is modified, a notification is sent to all satellites, including the Community application's production and management systems. Figure 6–2 shows the process flow.

> **Note:** Notifications are broadcasted in a synchronous manner, and therefore, a modification request is not completed unless all satellites acknowledge receiving this invalidation.

*Figure 6–2 Communication between WebCenter Sites and Community Application's Satellite Servers*
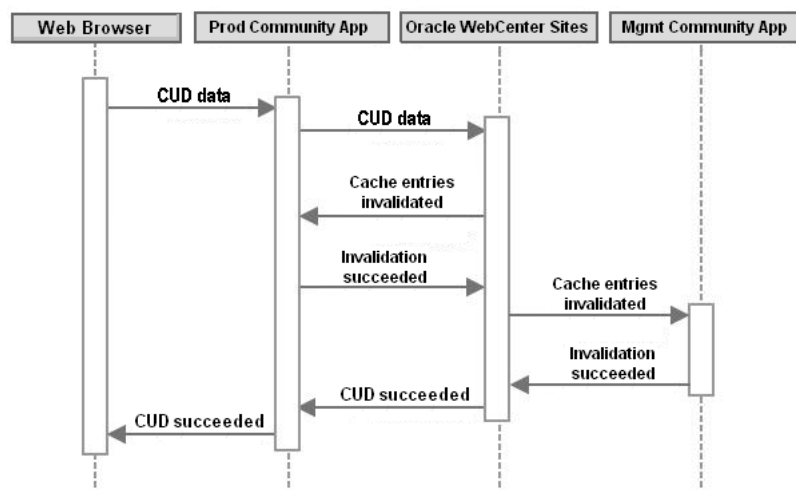
As shown in the process flow, the Community application caches results for all requests locally in the memory and receives invalidations to keep its cache consistent. For example, if the Community application collects, stores, and sends results for a request, and then a similar request comes, the application responds with cached data. However, if a visitor makes any changes between the time frame of the first and second requests, and an invalidation request is made in the same time frame, then the second request is served from the WebCenter Sites repository and not from the cache. (See also, Figure 6–3, Figure 6–4, and Figure 6–5)

The two caching modes of the Community application are: *regular caching* and *stale caching*. Both modes share the same approach to data reads. If the required data is in the cache, it is served immediately. When the data is not in the cache, it is loaded from the WebCenter Sites repository synchronously. Therefore, results in the user interface are displayed for the user only after the `load` operation is completed and the data is cached. However, data invalidations and cache consistency work differently in regular caching and state caching.

6.1.1.1 Regular Caching

When the Community application receives an invalidation request for a particular piece of data, it invalidates the cached data immediately so that all subsequent requests for that data get cache misses and the fresh data is loaded from the WebCenter Sites repository.

Figure 6–3 shows the process flow for the cache hit and cache miss events in the regular caching mode.

Figure 6–3   Cache Hit and Cache Miss in Regular Caching
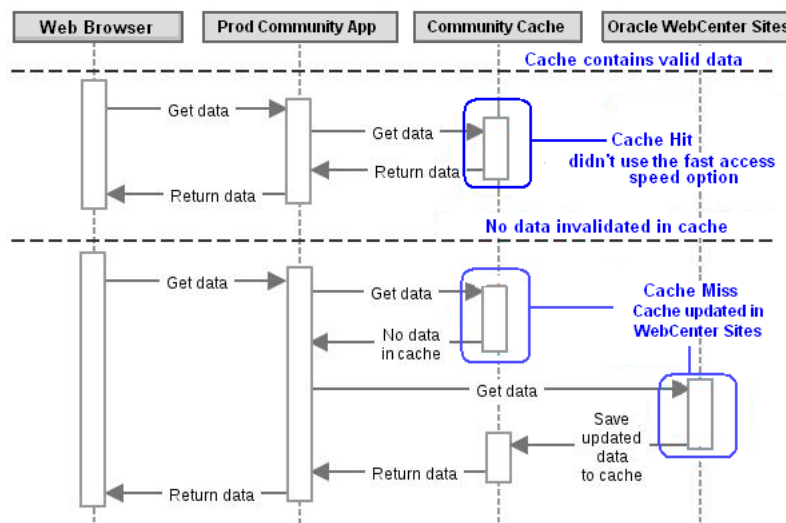
In the regular caching mode the data consistency level is high, and all the changes are immediately reflected in the interface. For example, when a user posts a new comment and refreshes the page, the new comment is shown on the page immediately. Similarly, content deleted from a page disappears immediately after the page is refreshed.

6.1.1.2 Stale Caching

When the Community application receives an invalidation request, it does not invalidate the data and purge it immediately. This is in contrast with regular caching in which the cached data is invalidated immediately. In the stale caching mode, the Community application just marks the data that is no longer valid with a special invalidation mark. However, the data still remains in the cache so it can still be served from cache memory. When a request for the invalidated data is received, a thread is launched to update the invalidated data. This process is called *regeneration cycle*.

Figure 6–4 shows how invalidated data is regenerated in the background and regenerated content is served from cache.

Figure 6–4   Regeneration Cycle in Stale Caching
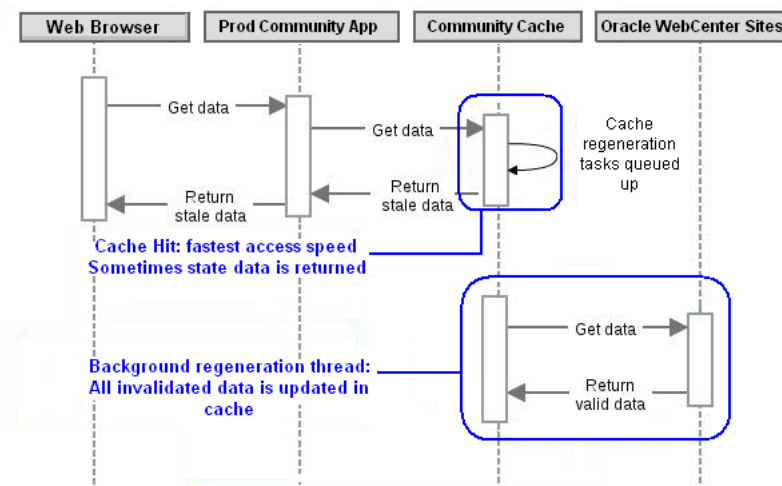
In the stale caching mode, the performance level is high because data loads happen seamlessly in the background. However, due to background data loads, the data consistency level is lower than that in the regular caching mode, so visitors do not see the actual results immediately. For example, when a comment is posted, it takes a couple of page refreshes before the comment is displayed on the page. The following process explains what happens in the stale caching mode:

1. Visitor posts a comment => the Community application marks the comment list with a special invalidation mark.

2. Visitor refreshes the page for the first time => the Community application still serves old data without the new comment from the cache, but detects the special mark and starts the background update process.

3. Visitor refreshes the page for the second time => the background update process is completed, the cache is updated => the new comment list containing the newly posted entry is returned to the client.

   The background update process starts with the first refresh. The second refresh displays the updated data because cache is updated by the end of the first refresh. too. However, it depends on network overheads and the volume of changes, so potentially it may take more than two refreshes.

By default, the Community application uses a regular caching schema. To boost performance, you can enable the stale caching in the `cos.war/WEB-INF/classes/wsdk_facilities.properties` configuration file by setting the `app_cache.stale` property to `true`, as shown in Example 6–1.

Example 6–1   Stale Cache Mode Enabled

```
#
# Application cache facility configuration
# + stale parameter enables usage 'old' data on production side
#
app_cache.stale=false
```

When the `app_cache.stale` property is set to `false`, the regular caching schema is used. However, certain areas in the product always need a high level of consistency, and therefore, the Community application overrides the stale cache mode for the following areas:

- The Community interface. Data needs to be always consistent with production systemfor easy management.

- Operations on visitors' profiles. Especially during registration when the application needs to determine if a user with such user name already exists.

You can manage cache through the Cache Management Console, which is accessible at `http://<cos_ip>:<cos_port>/<cos_context>/cache` on each Community application instance. For example, `http://mycosprod.us.oracle.com:8180/cos/cache/`. For information about cache management, see the chapter "Working with the Cache Tool" in the *Oracle WebCenter Sites User's Guide for the Community Application*.
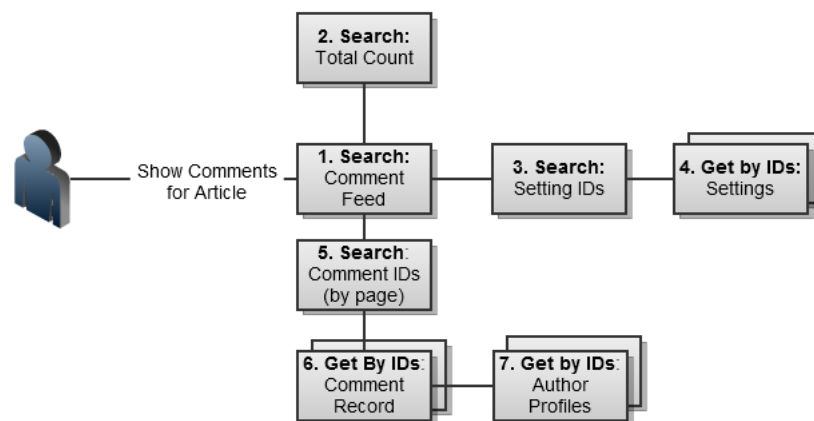
6.1.1.3 Caching Dependencies

The following types of dependencies are associated with caching:

- **Dependencies for content modification**: This type of dependency is recorded in the "Dependencies" column of the cache entries table such as "Commons Cache", and it contains the ID of the existing asset. The system invalidates this cache entry whenever the associated asset changes.

- **Dependencies for content creation**: This type of dependency is system-specific. The `unknowndeps: <table name>` type of the inCache dependency keyword is used in such cases. Whenever a new entry is posted in the table, the cache is invalidated. This type of caching is useful for search and count queries that are required to be invalidated if any new content is posted.

During cache management, user operations must be constantly mapped to the Community application logic and entries stored in the cache. For example, during comment loading, the data structures shown in Figure 6–5 are associated with the "Loading..." message (the Loading operation) which is displayed initially when the Comments widget starts loading.

Figure 6–5   Data Structure Associated with the Comments Operation

The following is a step-by-step explanation of Figure 6–5. These points describe how each operation for comments is performed and cached:

1. The CommentFeed object that aggregates the whole discussion is discovered.

2. The total number of comments is discovered. This number is used to calculate and render the pagination interface and other comment-related features.

3. The site settings configured in the Community application are loaded. These settings are applied to the CommentFeed object that is rendered on the page accessed by visitors. These settings determine where the "Post Comment" form will be shown (top or bottom) on a page, the level of commenting permissions, and so on. After the setting IDs are discovered, the Community application searches the cache and gets cache hits before contacting the database.

4. If site settings are not cached, they are fetched from the WebCenter Sites repository by their IDs. Lookup by ID is used whenever possible because of the effectiveness of the process.

5. Loading of actual comments to be displayed on the web site begins. The Comment Feed objects and CommentRecord objects have a one-to-many relationship. First, all the IDs of comments associated with a particular feed are discovered. After the IDs of the comments associated with a particular feed are discovered, the IDs are analyzed and validated against the local cache entries. The IDs that are missing in the cache are loaded with the next request.

6. CommentRecord objects are loaded from the WebCenter Sites repository by the IDs discovered on the previous step.

7. The comments are examined, and the author IDs are extracted from comments that were posted by authenticated users. Then, the visitor profile IDs for which there are cache hits are loaded from the WebCenter Sites repository.

6.2 Configuring Cache in the Community Application

The cache configuration in the Community application is similar to the cache configuration in WebCenter Sites because both products use the same inCache infrastructure.

The cache configuration file, `cos-cache.xml`, is located in the `cos-standalone-config` directory for each Community node directory.

For more information about configuration parameters in this file, see the chapter "Configuring inCache for Page Caching" in the *Oracle WebCenter Sites Administrator's Guide*.

6.3 Optimizing User-Generated Content (UGC) in the Community Application

There are certain peak hours when the visitor traffic and the amount of feedback are highest in a day. In other words, there is a time during a day when the maximum number of UGC entries are submitted to database. During these periods of peak loads, database cannot handle the content insertion rate at the speed at which visitors submit content entries. To address such situations, WebCenter Sites provides an option called *delayed writes*. This is a simple but effective tool that uses the producer-consumer pattern. Visitors submit content to the system in the form of comments, reviews, and so on. The content entries are then queued up for the database. A consumer thread picks up these entries from the queue and inserts them into the database one-by-one, so results of these incoming requests (UGC entries) display on the web site later. Since changes are buffered and applied in the background, there is data inconsistency, so users cannot see their changes immediately.

The delayed writes option is available only after the administrator has configured it in WebCenter Sites. For information about how to enable this option, see the "Buffering" chapter in the *WebCenter Sites: WEM Framework Developer's Guide*.

After the delayed writes option has been configured on the WebCenter Sites instance used by the Community application, you must also set this option in the application's `setup_cs.properties` configuration file located in the `cos.war/WEB-INF/classes` directory. In this file, set the `widgets.cs.production.attrs.delayed_writes` property to `true`:

```
#
# Enabling "delayed writes" mechanism. CS will persist the assets asynchronously.
# Default is "false"
#
widgets.cs.production.attrs.delayed_writes=true
```

Once the delayed writes option in enabled in the Community application, restart the Community production and management application servers. You will notice that the behavior of the production system will change when visitors submit content in the form of reviews, comments, and so on. For example, when a visitor posts a comment, the "Thank you for your submission" message will be displayed and comment will not appear on the site unless it is processed on the server asynchronously.

A

Guidelines for Maintaining the Community Application

This appendix includes the following sections:

- Widget Deployment Guidelines
- Adjusting Logging Levels
- Reporting Issues

A.1 Widget Deployment Guidelines

The deployment process can be started for the Community application widgets as soon as the widget functionality, permissions, and appearance are configured in the Community application. During widget deployment, when the "site settings" option is selected, configurations made in the Community application are instantly reflected on the deployed widgets, after the page containing the widget is reloaded. This option enables the deployed widget to constantly monitor configuration changes in the database. However, in the "custom settings" option, some of the configuration settings use custom values that are embedded into the widget deployment code snippet itself. Note that custom configuration settings are stored in the HTML/JavaScript code of the page on which the widget is deployed, and not in the database. As a result, custom configuration settings take priority over site settings.

When you deploy a widget with the "custom settings" option selected, the "Resource ID" field becomes available for customization. The resource ID enables the Community application to associate the content posted by visitors with the page on which it is deployed. Because of the lightweight integration approach — which uses JavaScript to deploy widget functionality to the web site — this ID is used to determine the type of content to be shown on a page. For example, if no resource ID is specified during deployment, md5 (a cryptographic hash function that produces a 128-bit (16-byte) hash value) is used as the resource ID automatically. In this case, when visitors visit a page repeatedly, the same content is shown to them by the Community application. Therefore, it is recommended that the resource ID be equal to the page ID (if it is managed by a content management system), so the association is straightforward and intuitive.

When the same comment feed needs to be shown on two separate pages, it is sufficient to specify the same resource ID. For example, xyz can be used as the resource ID on both pages, so that the same content is rendered on these pages.

A.2 Adjusting Logging Levels

This section includes the following:

- [Section A.2.1, "Configuring log4j Loggers"](#)
- [Section A.2.2, "Enabling Logging for SEO Widget JAR Files"](#)

A.2.1 Configuring log4j Loggers

Configuration and monitoring of log files is an important aspect of product maintenance. This section describes log4j loggers that are available for troubleshooting/monitoring.

- The `log4j-cos.properties` is the main configuration file for the logging in the system. This file is located in the `<Community_ Application>\deploy\management\management_node1\` and `<Community_ Application>\deploy\production\production_node1\` directories.

- To enable all the Community application-specific logging, set the root cos logger to debug level: `log4j.logger.com.fatwire.cos = DEBUG`.

- To monitor all the requests that the Community application sends to the WebCenter Sites data repository, enable the following logger:
  `log4j.logger.com.fatwire.cos.core.jpa.cmd.wem.WemCommandManager = TRACE`

- To monitor lifecycle management of CAS tickets that are used to connect to WebCenter Sites over REST, enable the following loggers:
  `log4j.logger.com.fatwire.cos.core.jpa.session.wem. WemSessionManager = TRACE`

  `log4j.logger.com.fatwire.cos.core.sites.wem.WemManager = TRACE`

- To monitor inCache operations performed by the Community application, enable the following logger:
  `log4j.logger.com.fatwire.cos.core.cache.appcache.wem.WemAppCacheFacilit y = TRACE`

- To monitor inCache invalidations coming from WebCenter Sites, enable the following logger:
  `log4j.logger.com.fatwire.cos.core.cache.appcache.wem.IncacheServlet = TRACE`

- When troubleshooting data consistency issues and the synchronization functionality (locking) in the system, enable the following loggers:
  `log4j.logger.com.fatwire.cos.cluster.CacheLockClientFacility = TRACE`

  `log4j.logger.com.fatwire.cos.cluster.ClusterLockImpl = TRACE`

A.2.2 Enabling Logging for SEO Widget JAR Files

When deploying widgets that support search engine optimization (SEO), you need to download the `cos-widget-tag.jar` file containing the widget deployment and rendering logic, and place it into the classpath of your web applications. Navigate to **Comments** > **Deploy** > **Comments**, then select **Custom settings** and scroll down to the "Widget Tag" field. To download the JAR file, click the link in the "Note" section.

This is a lightweight JAR file, and it is created to function independently of any external logging libraries. However, to enable log messages provided by this library (`cos-widget-tag.jar`), it is necessary to add the `-Dcos.widget.tag.debug=true` Java parameter to JVM that runs the web application. This ensures that the log messages display in the standard output stream (console).

To optimize this library for performance, it is recommended that you add the following JVM parameters:

```
-Dhttp.keepAlive=true, -Dhttp.maxConnections=<Number>
```

The `<Number>` value must be aligned with the concurrency rates on the application server on which the pages with the SEO deployment tag were deployed, as well as with the number of processing threads available on the application server.

A.3 Reporting Issues

Before contacting the support team, compile the information required for investigation. This will help you avoid unnecessary communication round-trips and speed up the troubleshooting process.

The following is the recommended list of items to be provided to support for quick and efficient troubleshooting:

Environment Details

- Operating system name and version

- System language, locale, and time zone

- JVM name and version

- Application server name and version

Configuration Details

Archive the contents of the product configuration directory and attach the archive to the support ticket. For example, `Production_node1` or `Management_Node1`, configurations are saved in the `<install folder>/deploy/production` and `<install folder>/deploy/management` directories. Other required files are: the `setup_*.properties` files from the `cos.war/WEB-INF/classes` directory.

Logging Details

- Log files of the Community application and its CAS

- Log files of the WebCenter Sites application and its CAS

- Log files of the Community application and WebCenter Sites application servers (including the standard output and standard error consoles)

Index

W

WebSphere Application Server (WAS)
    enabling social networking services,   2-14
        export security certificate from Facebook,   2-15
        export security certificate from Janrain,   2-17
        export security certificate from Twitter,   2-16
        import security certificates,   2-18
widget deployment guidelines,   A-1