# Oracle® Application Server Syndication Services

Developer's and Administrator's Guide

10*g* (9.0.4)

**Part No. B10667-01**

September 2003

Oracle Application Server Syndication Services manages and automates the establishment of syndication relationships (offers and subscriptions), content transfers based on delivery rules (contracts), and results analysis (access log).

ORACLE®

Oracle Application Server Syndication Services Developer’s and Administrator’s Guide, 10*g* (9.0.4)

Part No. B10667-01

Primary Author: Rod Ward and Marco Carrer

Contributing Author: Cheng Han, Xiaohua Lu, Giuseppe Panciera, Alok Srivastava, Timothy Chien, Susan Shepard, Deborah Owens

# Contents

## List of Examples

## List of Figures

## List of Tables

# Send Us Your Comments

**Oracle Application Server Syndication Services Developer's and Administrator's Guide, 10*g* (9.0.4)**

**Part No. B10667-01**

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most about this manual?

If you find any errors or have any other suggestions for improvement, please indicate the title and part number of the documentation and the chapter, section, and page number (if available). You can send comments to us in the following ways:

- Electronic mail: nedc-doc_us@oracle.com
- FAX: 603.897.3825   Attn:  Oracle Application Server Syndication Services Documentation
- Postal service:
  Oracle Corporation
  Oracle Application Server Syndication Services Documentation
  One Oracle Drive
  Nashua, NH 03062
  USA

If you would like a reply, please give your name, address, telephone number, and electronic mail address (optional).

If you have problems with the software, please contact your local Oracle Support Services.

# Preface

As a feature of Oracle Application Server, Oracle Application Server Syndication Services ("Syndication Services") manages and automates the establishment of syndication relationships (offers and subscriptions), content transfers based on delivery rules (contracts), and results analysis (access log).

## Audience

This guide is for developers who want to automate the establishment of syndication relationships and content transfers based on delivery rules.

This guide is also for administrators who will be managing Syndication Services and monitoring results analysis.

## Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Standards will continue to evolve over time, and Oracle Corporation is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For additional information, visit the Oracle Accessibility Program Web site at `http://www.oracle.com/accessibility/`.

**Accessibility of Code Examples in Documentation** JAWS, a Windows screen reader, may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, JAWS may not always read a line of text that consists solely of a bracket or brace.

**Accessibility of Links to External Web Sites in Documentation** This documentation may contain links to Web sites of other companies or organizations that Oracle Corporation does not own or control. Oracle Corporation neither evaluates nor makes any representations regarding the accessibility of these Web sites.

## Organization

This guide contains the following chapters and appendixes:

### Chapter 1, "Introduction to Syndication Services"

This chapter introduces Oracle Application Server Syndication Services, explains terminology and concepts, and provides an overview of administration and user tasks.

### Chapter 2, "Administration of Syndication Services"

This chapter describes in detail Oracle Application Server Syndication Services administration tasks.

### Chapter 3, "Client Application Development"

This chapter describes client application development and content provider development and registration.

### Chapter 4, "Content Connector Development"

This chapter describes content connector development and installation.

### Appendix A, "Error Messages"

This appendix describes Oracle Application Server Syndication Services error messages.

### Appendix B, "Syndication Services Security"

This appendix describes Oracle Application Server Syndication Services security information.

### Appendix C, "Sample Syndication Client"

This appendix provides a listing of the `SampleSyndicationClient.java` program.

### Appendix D, "RSS Content Connector (CPAdaptor)"

This appendix provides a listing of the `RSSCPAdaptor.java` program.

## Related Documentation

For more information about using and managing Syndication Services, see the following documents in the Oracle documentation set:

- *Oracle9i Java Developer's Guide*
- *Oracle Application Server Web Services Developer's Guide*

For reference information in Javadoc format, see the Oracle API documentation (also known as Javadoc). The API documentation can be found on the Oracle Application Server 10*g* Documentation Library CD-ROM as Syndication Services Client API Reference (Javadoc) under Oracle Application Server Syndication Services, which is located under the Portals tab.

Printed documentation is available for sale in the Oracle Store at

```
http://oraclestore.oracle.com/
```

To download free release notes, installation documentation, white papers, or other collateral, please visit the Oracle Technology Network (OTN). You must register online before using OTN; registration is free and can be done at

```
http://otn.oracle.com/membership/
```

If you already have a user name and password for OTN, then you can go directly to the documentation section of the OTN Web site at

```
http://otn.oracle.com/documentation/
```

## Conventions

Oracle Application Server Syndication Services is also referred to as "Syndication Services" in this guide.

The following conventions are also used in this guide:

| Convention | Meaning |
|---|---|
| .<br>.<br>. | Vertical ellipsis points in an example mean that information not directly related to the example has been omitted. |
| . . . | Horizontal ellipsis points in statements or commands mean that parts of the statement or command not directly related to the example have been omitted |
| **boldface text** | Boldface text indicates a term defined in the text. It is also used to denote GUI elements. |
| *italic text* | Italic text is used for emphasis and for book titles. |
| < > | Angle brackets enclose user-supplied names. |
| [ ] | Brackets enclose optional clauses from which you can choose one or none. |

1

# Introduction to Syndication Services

Oracle Application Server Syndication Services ("Syndication Services") manages and automates the establishment of syndication relationships (offers and subscriptions), content transfers based on delivery rules (contracts), and results analysis (access log). Administrators can also manage Syndication Services properties.

Section 1.1 introduces some Syndication Services terminology and concepts, and Section 1.2 describes some simple usage flow charts that describe the process of administering Syndication Services from offer creation to subscription management.

## 1.1 Terminology and Concepts

Syndication Services content provider terminology includes:

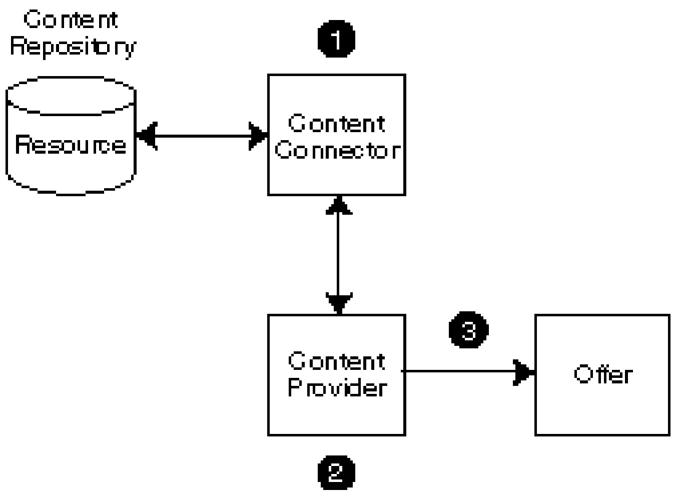
- Content -- the information of interest to users that is delivered as a package.
- Syndication -- the delivery of content from an external content repository to subscribed users.
- Syndicator -- the content distributor, in this case Syndication Services.
- Administrator -- someone who administers Syndication Services by performing tasks, such as creating and managing content providers, offers, and contracts, as well as granting users or groups access to an offer, and managing subscriptions.
- User -- someone who is allowed to perform a task such as subscribing to an offer; also known as a *subscriber*.
- Content provider -- the entity that provides content to the syndicator (Syndication Services) for syndication by means of a content connector. The content provider represents the external content repositories.

- Content connector -- the software components used by Syndication Services to interact with external content repositories, also known as content provider adaptors. Syndication Services provides connectors, such as file systems or Web distributed authoring and versioning (WebDAV) folders. See Chapter 4 for more information about developing a content connector. Through the content connector, the administrator can specify the parameters needed to connect to the content repository.
- Resource -- the specific location of the external content repository to be syndicated. For a file system content connector, the resource is a directory under the root directory. For a WebDAV content connector, the resource is a folder under the base folder. Each content provider identifies one resource and the set of parameters needed to access that resource.
- Offer -- the smallest unit of content to which users can subscribe. An offer can be created for each content provider resource. The resource, for example, can be a directory in the file system or a folder accessible through WebDAV. Each offer is associated with a contract that contains the business terms, expiration policy, and delivery rules.

Figure 1–1 shows that the content provider uses its content connector to expose resources from the content repository to the administrator, who then creates an offer for each resource to which a user can be granted access and can subscribe. The administrator performs these tasks:

1. Selects the content connector when creating the content provider.
2. Creates and manages content providers.
3. Creates an offer for each content provider resource.

***Figure 1–1 Content Provider Provides Content and the Administrator Creates an Offer for Each Content Provider Resource***



Syndication Services syndication relationship terminology includes:

- Contract -- associated with each offer, it contains the business terms, expiration policy, and delivery rules for delivering content to a subscribed user.
- Business terms -- a detailed description of the terms of agreement for a contract associated with an offer.
- Expiration policy -- describes when subscriptions based on a contract will expire.
- Delivery rules -- describes in detail how the content is to be delivered to subscribed users, when it will be delivered, and how often it will be delivered.
- Subscriptions -- an agreement between the user (subscriber) and the syndicator for the delivery of content associated with an offer.

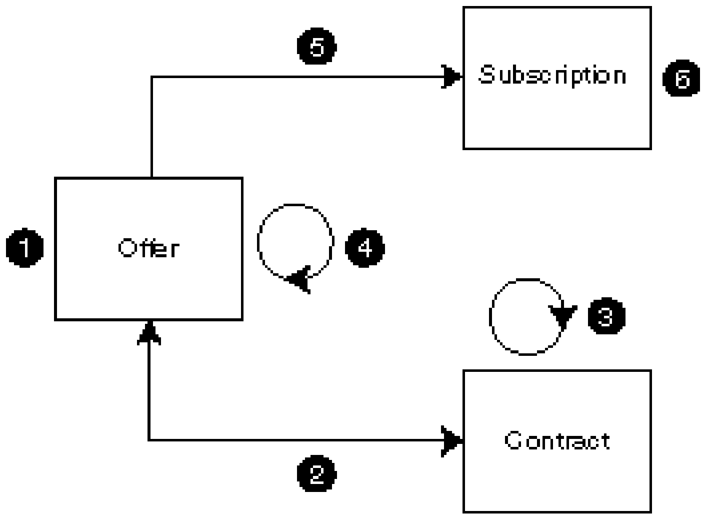
Figure 1–2 shows that users or groups of users are given access to an offer with a contract by the administrator, who manages each offer and its contract. When the user subscribes to the offer, the relationship becomes a subscription. Content will be delivered to the user as a package for a set period of time, based on the offer’s expiration policy, and in a manner defined by prescribed delivery rules.

The administrator and the user (subscriber) perform these tasks:

1. The administrator creates and manages offers.

2. The administrator grants users or groups access to an offer with a contract.
3. The administrator creates and manages the contract and its associated offer.
4. The user (subscriber) accesses a catalog of offers.
5. The user subscribes to an offer.
6. The administrator manages subscriptions.

***Figure 1–2 A User (Subscriber) Subscribes to an Offer with a Contract to Establish a Subscription***



Section 1.2 describes some simple usage flow charts that illustrate the sequence of tasks that a Syndication Services administrator performs in managing a syndication system, from offer creation to subscription management.

## 1.2 Usage Flow Charts

Usage flow charts illustrate the sequence of steps and information flow among the Syndication Services components involving the administrator of the syndication system, as shown in Figure 1–1 and Figure 1–2. This section describes this in more detail. This includes creating and managing content providers, granting users or groups of users access to offers to which they want to subscribe, contract creation and management, and finally, the management of subscriptions belonging to subscribed users. Section 1.2.1 and Section 1.2.2 describe some of these usage flow charts from a Syndication Services administrator perspective.

### 1.2.1 Administrator Usage Flow Charts

The Syndication Services administrator performs the following tasks when initially setting up a syndication system:

1. Registers each content provider that identifies the new content resources to be syndicated.
    a. Enters the content provider properties, such as its name, the e-mail address of the contact person owning the content, and a brief description of the content provider.
    b. Selects a content connector from the list of content connectors as the means to identify the connector needed for the desired content repository, such as FileConnector or WebDAVConnector.
    c. Specifies the settings used to access the content repository. These settings will vary according to the type of content connector chosen.

       For example, a FileConnector content connector uses a file system as its content repository, and its File Root Path setting defines the base of the content. Every directory under the root directory is considered a content provider resource for which the administrator can create an offer. A WebDAV content connector uses folders as its content repository, and its DAV Root Content setting defines the base folder for the content. Every folder under the base folder is a content provider resource for which an administrator can create an offer.
    d. Ensures that the content resource is located in the specified place, such as in the identified directory under the specified root directory of the file system, or in the identified folder under the specified WebDAV base folder.
2. Creates an offer for each identified content resource to be syndicated.
    a. Selects the content provider resource with which the administrator creates an offer.
    b. Enters a brief description of the offer and specifies the content usage properties, such as intellectual property rights and requested content usage patterns.

   c. Selects the users and groups to which the administrator grants subscribe privilege to this offer along with the contract to associate with these users. Note that the administrator can also create a contract specifically for this offer for this user. To associate a special contract with this offer, the contract must already have been created along with the contract business terms, so the administrator can then select the contract in this step.

3. Manages the subscriptions that are made between users (subscribers) and the syndicator. See Section 1.2.2 for more information on managing subscriptions.

As additional content resources become available, such as additional directories under a content provider's file system root directory or additional folders under a content provider's WebDAV base folder, the administrator must create an offer for each new resource.

As other content resources are identified besides file systems and WebDAV folders, specialized content connectors must be created to identify each of these new resources. See Chapter 4 for more information about building a content connector and installing it. Once the specialized content connector is built and installed, then the administrator again performs Steps 1 through 3.

With this overview of how the administration tasks flow, the administrator can begin setting up the syndication system. The first tasks to perform are described in Section 2.2. These tasks involve specifying the initial set of parameters for the property groups: general, scheduler configuration, and HTTP/S and SMTP transport properties. In addition, the administrator should review the access log and purge access log entries. The access log records user access to Syndication Services, as described in Section 2.7.

### 1.2.2 Subscription Usage Flow Charts

Once a subscription is created between the user (subscriber) and the syndicator for a particular offer, the Syndication Services administrator will need to manage the subscription. These tasks include:

1. Reviewing subscriptions. If the state of a subscription is Active, the administrator can change the state to Suspended or Terminated if that subscription must be suspended or terminated. A suspended subscription can be reactivated to the Active state. However, a terminated subscription can only be purged.
2. Purging expired and terminated subscriptions.
3. Viewing other subscription information, such as confirmation notification, expiration policy, and delivery rules. The administrator can review the accepted

offer information associated with the subscription, including content usage properties, which includes intellectual property rights, requested content usage patterns, and content provider and resource information.

# 2 Administration of Syndication Services

Syndication Services administration includes the following tasks:

- Completing a Postinstallation Configuration Task
- Setting Syndication Services Properties
- Understanding User Management
- Registering and Managing Content Providers
- Creating and Managing Offers
- Managing Subscriptions
- Reviewing and Purging an Access Log
- Monitoring Performance and Tuning
- Performing Advanced Administration Tasks

The sections that follow describe each of these administration tasks.

## 2.1 Completing a Postinstallation Configuration Task

Before you can begin using Syndication Services, you must set the Domain property, which is the domain name of the system where the Oracle Application Server infrastructure database is installed. See Section 2.2 for more information about setting this property.

## 2.2 Setting Syndication Services Properties

Before you can begin registering content providers and offers, managing subscriptions, and syndicating content, you must first set Syndication Services properties. At the **Oracle Enterprise Manager Syndication Services Administration** page, click **System Properties**. The **System Properties** page is displayed.

The properties on this page that must be set are: access logging, domain, scheduler status, HTTP/S transport, and SMTP transport. Table 2–1 describes all the Syndication Services properties and provides guidelines for the properties that must be set.

> **Note:** If you set any Syndication Services properties, a restart of the OC4J_Portal instance where the Syndication Services application is deployed is required for these settings to take effect.

***Table 2–1 Syndication Services Properties***

| Property | Description |
|---|---|
| **General Properties** | |
| Instance ID | Specifies the generated ID for this Syndication Services instance. |
| Instance Name | Specifies the name of this Syndication Services instance. |
| User Agent Name | Specifies the name of the user agent associated with this Syndication Services instance and is analogous to the HTTP User Agent name. |
| Access Logging *(Must be set.)* | Enables or disables the collection of user access information. Set this to **On** if you want to enable the collection of user access information. Access logging records each user access to Syndication Services as an entry in the access log, which you can review and then purge as needed. |
| Domain *(Must be set.)* | Specifies the domain name of the system where the Oracle Application Server infrastructure database is installed, using the following format, for example: `us.oracle.com`. The domain name is required by Syndication Services Administration pages when interacting with Oracle Delegated Administration Services pages for selecting users and groups of users, and granting them the subscribe privilege to offers. |
| **Scheduler Configuration Properties** | |
| Scheduler Status *(Must be set.)* | Indicates whether or not the scheduler is enabled to allow automatic push deliveries to occur. Set scheduler status to **On** to allow push deliveries to occur. See Section 2.9.4 for more information. |

*Table 2–1 Syndication Services Properties (Cont.)*

| Property | Description |
|---|---|
| Timer Pool Size | Specifies the number of concurrently active timers to be used by the scheduler. The default value is 16 timers. See Section 2.9.4 for more information. |
| Monitor Frequency (msec) | Specifies how frequently the scheduler checks for new subscriptions that require content to be pushed. The default value is 60000 msec or 1 minute. See Section 2.9.4 for more information. |
| **HTTP/S Transport Properties** *(Must be set.)* | |
| Proxy Host | Specifies the name of the proxy server to perform content pushes to servers outside the firewall. |
| Proxy Port | Specifies the port number for the HTTP proxy host. |
| Connection Timeout | Specifies the length in seconds that Syndication Services will wait to establish a connection to the content source or repository before it returns a timeout message. |
| Wallet Location | Specifies the location of Oracle Wallet Manager required to establish HTTPS connections for content pushes. See "Using Oracle Wallet Manager" in *Oracle Advanced Security Administrator's Guide* for more information. |
| **SMTP Transport Properties** *(Must be set.)* | |
| SMTP Server | Specifies the name of the SMTP server. |
| From | Specifies the e-mail address of the sender to be used when performing content pushes through e-mail. |
| Subject | Specifies the subject portion of the e-mail header to be used when performing content pushes through e-mail. |

## 2.3 Understanding User Management

Oracle Application Server Syndication Services, 10*g* (9.0.4) uses Oracle Internet Directory (OID) of the Oracle Application Server infrastructure as the default user repository. This is achieved through the use of lightweight directory access protocol (LDAP)-based provider of Oracle Application Server Containers for J2EE (OC4J) Oracle Application Server Java Authentication and Authorization Service (JAAS).

Syndication Services specific OID groups are located under the `cn=syndication_groups` subtree of the group subtree of the OID default subscriber. Syndication

Services specific users are located under the user subtree of the OID default subscriber.

The types of Syndication Services groups are summarized in Table 2–2.

***Table 2–2 Default Syndication Services Groups***

| Group Name | Description |
|---|---|
| syndsubscribers | Users under this group can ping the server, retrieve a catalog, create or change subscriptions, get events (activity associated with a particular subscription), retrieve subscription status, pull content related to subscriptions, and cancel subscriptions. |
| syndcps | Users under this group can notify the server of any content updates on the content provider side. See Section 4.1.7 for more information about event push support. |
| syndschedulers | This is an internal group defined by Syndication Services implementation. Administrators must not edit, add users to, or delete users from this group. This group performs push operations based on the subscription delivery rules. |

**Note:** Do not remove any of these default Syndication Services groups.

By default, the following users are created in an installation. Administrators can add or remove users to or from these corresponding groups as shown in Table 2–3.

***Table 2–3 Default Syndication Services Users***

| Group Name | User Names | Comments |
|---|---|---|
| syndsubscribers | syndication | The sample subscribers. Administrators can add new users to this group. |
| | portal_syndication | The portal_syndication user is primarily used for portal integration. This user must not be removed. |
| | uddi_syndication | The uddi_syndication user is primarily used for uddi integration. This user must not be removed. |
| syndcps | sample_cps | Sample content provider user. This user must not be removed. |
| syndschedulers | syndscheduler | The default scheduler user performs push operations. This user must not be removed. |

Generic user management, such as creation, deletion, suspension, and so forth, is handled by OID and its Delegated Administration Service (DAS). Refer to *Oracle Internet Directory Administrator's Guide* for more information.

---

**Note:** In DAS, the display name may be different from the actual name.

---

To grant Syndication Services access to existing OID users, use DAS to assign each user to the syndsubscribers group.

User management, including operations such as creation, deletion, suspension, role management, and so forth, is handled by OC4J Java Authentication and Authorization (JAAS) service. Refer to *Oracle Application Server Containers for J2EE Services Guide* for more information.

## 2.4 Registering and Managing Content Providers

Before you can create any offers to which users can subscribe, you must first register content providers that identify where the content exists. A content provider relies on a content connector, which is the software interface that is used for connecting to an external content repository and its resources. Content provider settings identify the specific location of the repository and any other essential characteristics of the repository. An offer can be created for each identified resource in the external content repository. Content providers denoted as available means that the class for that content provider has been found in the classpath.

For example, a file connector uses a file system as its content repository. The **File Root Path** setting defines the root location of the content. Every directory under the root directory is a content provider resource for which you can create an offer.

As another example, a WebDAV connector uses folders as its content repository. The **DAV Root Content** setting defines the base folder for the content. Every folder under the base folder is a content provider resource for which you can create an offer.

At the **Oracle Enterprise Manager Syndication Services Administration** page, click **Content Providers**. The **Content Provider Management** page is displayed.

At the **Content Provider Management** page, you can:

- Review Content Connector Information
- Register Content Providers

- Edit Content Provider Information
- Delete Content Providers

## 2.4.1 Review Content Connector Information

To review the list of available content connectors, click the **Content Connectors** tab. A **Content Connectors** page is displayed. You can review the list of available content connectors. Content connectors denoted as available means that the class for that content connector has been found in the classpath.

To view the properties of a content connector, either click the name of the content connector in the **Name** column, or click its radio button in the **Select** column and then click **View**. A **Content Connectors** page is displayed where you can review the name of the content connector, the description of the content connector, and the name of its Java class. When you are through reviewing this information, click **OK** to return the **Content Connectors** page.

### 2.4.1.1 Built-in Connectors

Syndication Services comes configured with the following content connectors:

- FileConnector -- a connector that uses a file system as a content repository that defines a root directory as the root location of the content. Every directory under the root directory is considered to be a content provider resource for which an administrator can create an offer.
- WebDAVConnector -- a connector that uses the distributed authoring and versioning (DAV) protocol to create bindings to a content resource. Every folder under the base folder is considered to be a content provider resource for which an administrator can create an offer.
- UDDI connectors -- a set of nine preconfigured connectors from which an administrator can choose to create content provider resources, and subsequently create a set of offers. UDDI (Universal Discovery Description and Integration) publishers in the Oracle Application Server UDDI Registry can subscribe to these offers and monitor or obtain changes in the registry. For more information, see the UDDI Subscription Service information in *Oracle Application Server Web Services Developer's Guide*.

A Really Simple Syndication (RSS) sample content connector, including the full source file of this sample, can be downloaded from the Oracle Technology Network (OTN) Web site `http://otn.oracle.com/tech/java/oc4j/demos/`. This connector is described in Chapter 4, which takes a syndication feed (content updates) that an administrator can create for a content provider resource, from which an offer can be created, and to which users can subscribe. Appendix D provides a listing of the `RSSCPAdaptor.java` program that implements this connector.

Section 4.1 describes how to develop a new content connector and Section 4.2.1 describes how to install a new content connector.

### 2.4.2 Register Content Providers

To register a content provider, click **Register Content Provider**. A register content provider wizard displays the first **Register Content Provider: Properties** page.

1. At the **Register Content Provider: Properties** page, enter the required properties for this content provider, such as its name, e-mail address of the owner of the content provider, and a brief description of the content provider. Then, click **Next** to continue to the **Register Content Provider: Connector** page.
2. At the **Register Content Provider: Connector** page, select the content connector to be used to access this repository by selecting its radio button. Then, click **Next** to continue to the **Register Content Provider: Set Up** page.
3. At the **Register Content Provider: Set Up** page, configure the content connector properties by entering the values required for registration, then click **Finish** to create the content provider and return to the **Content Provider Management** page.

### 2.4.3 Edit Content Provider Information

To edit content provider information, click the content provider name in the **Name** column that you want to edit, or select the content provider by clicking its radio button in the **Select** column, and then click **Edit**. The **Edit Content Provider** page is displayed. On this page, you can:

- View the content provider name.
- View the type of content connector.
- Edit the e-mail address of the owner of the content provider.
- Edit the description of the content provider.
- Edit any content connector configuration settings.

### 2.4.4 Delete Content Providers

To delete a content provider, select the content provider by clicking its radio button in the **Select** column, and then click **Delete**. A warning message appears indicating that unregistering a content provider implies the deletion of all its offers. Click **Delete** to confirm.

**Note:** You cannot delete a content provider if there are active (not expired and terminated) subscriptions that depend on offers based on its resources. In order to delete a content provider, you have to terminate all those subscriptions.

## 2.5 Creating and Managing Offers

After you have registered some content providers, you can begin the process of creating offers to which users can subscribe.

At the **Oracle Enterprise Manager Syndication Services Administration** page, click **Offers**. The **Offer Management** page is displayed.

At the **Offer Management** page, you can:

- View the List of Content Providers
- View the Offers for a Content Provider
- Create an Offer for a Content Provider
- Manage Contracts and Business Terms

### 2.5.1 View the List of Content Providers

The list of content providers is displayed along with descriptions of the content providers.

### 2.5.2 View the Offers for a Content Provider

To view the offers for a content provider, select a content provider from the list by clicking its radio button in the **Select** column, then click **View Offers**. An **Offers by Content Provider** list page is displayed, which lists the offers for this content provider.

On the **Offers by Content Provider** list page, you can:

- Edit the Properties of an Offer

- Edit the Access List of Users and Groups Who Can Access the Offer
- Delete an Offer
- Create an Offer for a Content Provider

#### 2.5.2.1 Edit the Properties of an Offer

To edit the properties of an offer, select the offer by clicking its radio button in the **Select** column, then click **Edit**. An **Edit Offer** page is displayed where you can edit:

- **General properties** -- offer name, offer description, and the offer state: Active or Suspended.

> **Note:** Suspended offers will not be available for subscription.

- **Content Usage Properties** -- Information Content Exchange (ICE) 1.1 standard properties: product name to which this offer belongs, name of the holder of the intellectual property rights, content status of the intellectual property rights (see Table 2–4); and content usage patterns (see Table 2–5). See Table 2–11 for more information about these offer properties. Syndication Services does not enforce these properties; the subscriber client application must handle them.
- **Content Provider** -- name and resource name.

*Table 2–4 Intellectual Property-Rights Categories*

| Category | Description |
|---|---|
| Public Domain | Indicates the content has no licensing restrictions. |
| Free with Acknowledgment | Indicates the content has no licensing restrictions beyond a requirement to display an acknowledgment of the content source. |
| See License | Indicates the content has restrictions as agreed to in an existing licensing agreement. |
| Severe Restriction | Indicates the content has licensing restrictions that require special attention. |
| Confidential | Indicates the content is confidential and must be protected. |
| Other | Indicates the content has some other intellectual property-rights status not previously listed. |

***Table 2–5 Content Usage Pattern Categories***

| Category | Description |
|---|---|
| Atomic Use | Indicates that *all* the content to be delivered must be used, or *none* of it is to be used. The default is false (unchecked) and indicates that atomic use of the content is not required. |
| Editable | Indicates whether the user (subscriber) may modify the information or must use it exactly as it is delivered. The default value is false (unchecked), which indicates that the content cannot be edited, and therefore must be used exactly as delivered. |
| Show Credit | Indicates whether or not the user (subscriber) must prominently display attribution for the content source. The default value is false (unchecked), which indicates that the user is not required to acknowledge the source of the data. |
| Usage Required | Indicates whether or not the syndicator expects its users to return usage data regarding the intended or ultimate viewers of the syndicated content. |

#### 2.5.2.2 Edit the Access List of Users and Groups Who Can Access the Offer

To select the users and groups to whom you want to grant subscribe privilege to this offer, select users by clicking the user icon, then click the desired user names. To select a group click the group icon, then click the desired group names.

Generic user management, such as creation, deletion, suspension, and so forth, is handled by OID and DAS. See Section 2.3 for a list of default groups and users created at installation. Refer to *Oracle Internet Directory Administrator's Guide* for more information.

**Note:** The list of users and groups from which you can select are all the users and groups defined in OID. Remember that only users belonging to the `syndsubscribers` group can connect to Syndication Services to perform syndication operations. For more information, see Section 2.3.

If you use Netscape 7.0, selecting a user may result in no action; that is, the user is not selected. To work around this problem, use Internet Explorer 6.0 or later.

Next, you can change the contract you want to associate with the user and groups you selected by selecting the desired contract from the drop-down menu, and clicking **Add** to change it. You can revoke access to the offer for a user or group by clicking the delete icon for that specific user or particular group name. When you have finished granting or revoking users and groups access to this offer, click **OK** to save all changes and you will be returned to the **Offer Management Offers** page.

#### 2.5.2.3 Delete an Offer

To delete an offer, click its radio button in the **Select** column, then click **Delete**. A warning message is displayed. Click **Delete** to delete the offer.

#### 2.5.2.4 Create an Offer for a Content Provider

See Section 2.5.3 for more information.

## 2.5.3 Create an Offer for a Content Provider

To create an offer, click **Create Offer**. A create offer wizard displays the first **Create Offer: Resource** page.

1. At the **Create Offer: Resource** page, select the content provider resource for which you want to create an offer by clicking the flashlight and selecting the resource. Resources from a content provider become available after registering the content provider (see Section 2.4.2). Then, click **Next** to continue to the next create offer wizard page.

2. At the **Create Offer: Properties** page, specify the following information:

   – **General properties** -- offer name and offer description.

   – **Content Usage Properties** -- ICE 1.1 standard properties: name of the holder of the intellectual property rights, content status of the intellectual property rights (see Table 2–4): public domain, free with acknowledgment, see license, severe restriction, confidential, or other; content usage patterns (see Table 2–5): atomic use, editable, show credit, and usage required. These content usage properties are optional properties notifying users about possible restrictions associated with the usage of the offer content.

   Then, click **Next** to continue to the next create offer wizard page.

3. At the **Create Offer: Contracts** page, select the users and groups to whom you want to grant access to this offer, then choose the contract you want to associate with selected users and groups. Select users by clicking the user icon, then clicking the desired user names. Select groups by clicking the group icon, then clicking the desired group names. Change access to the offer for a user or group by clicking the delete icon for that particular user or group name. When you

have finished giving users and groups access to the offer, click **Finish** to create the offer.

---

**Note:** The list of users and groups from which you can select are all the users and groups defined in OID. Remember that only users belonging to the syndsubscribers group can connect to Syndication Services to perform syndication operations. For more information, see Section 2.3.

---

### 2.5.4 Manage Contracts and Business Terms

At the **Offer Management Contracts** page, you can:

- Edit a Contract
- Edit the Contract Business Terms
- Delete a Contract
- Create a Contract
- Create the Contract Business Terms

#### 2.5.4.1 Edit a Contract

To edit a contract, click its radio button in the **Select** column, then click **Edit**. An **Edit Contract General** page is displayed. On this page, you can edit:

- **General properties** -- contract name, contract description, and requires confirmation. Requires confirmation means whether or not confirmation of receipt of the content package is required before the next content package can be delivered to the user. A content package represents a single unit of content that is to be delivered.
- **Expiration policy** -- expiration information for subscriptions based on this contract (see Table 2–6), such as the start and stop dates, including the time zone, and maximum number of deliveries for the subscription, and the expiration priority (see Table 2–7): time-based, quantity-based, first, or last.

***Table 2–6 Expiration Policy Categories***

| Category | Description |
|---|---|
| Time Zone | Indicates a geographical region within which the same standard time is used, usually designated by three alphabetical characters, such as EST to represent Eastern (United States) Standard Time. |

*Table 2–6 Expiration Policy Categories (Cont.)*

| Category | Description |
|---|---|
| Start Date | Indicates the date that delivery of content will begin. By default, this means the beginning of the day (00:00:00 AM). |
| Expiration Priority | Specifies the expiration criteria. See Table 2–7 for a detailed explanation of the possible values. |
| Stop Date | Indicates the date that delivery of content will expire. By default, this means the end of the day (23:59:59 PM). To specify an unlimited expiration policy, specify a time-based expiration priority and do not specify a stop date. |
| Maximum Quantity | Indicates the maximum value, expressed as an integer of value 0 or greater, for the number of allowed deliveries. |

*Table 2–7 Expiration Priority Values*

| Value | Description |
|---|---|
| Time-Based | Indicates the subscription will expire when the stop date is reached. |
| Quantity-Based | Indicates the subscription will expire when the number of allowed deliveries is reached. |
| First | Indicates the subscription will expire when the stop date or the number of allowed deliveries is reached. |
| Last | Indicates the subscription will expire only when both the stop date and the number of allowed deliveries are reached. |

- **Pull delivery rules** -- **Enable Pull Delivery** selected: a set of controls delivers content to subscribed users, on demand. Pull delivery rules describe the daily time period during which content pulls are allowed (see Table 2–8), starting at the time specified and lasting for the duration specified. Specify the time zone, the start time in hours and minutes for pull content delivery, the total duration in hours and minutes for the delivery of content, and the days of the week or days of the month in which pull delivery is allowed.

*Table 2–8 Pull and Push Delivery Category Descriptions*

| Category | Description |
|---|---|
| Time Zone | Indicates a geographical region within which the same standard time is used, usually designated by three alphabetical characters, such as EST to represent Eastern (United States) Standard Time. |

*Table 2–8 Pull and Push Delivery Category Descriptions (Cont.)*

| Category | Description |
|---|---|
| Start Time | Specifies the time (hour and minute) and AM or PM when the pull content delivery is to start. The start time by default is the beginning of the day (00:00 AM). |
| Duration | Specifies the total time in hours and minutes allowed for the content to be delivered. This value must not exceed 24 hours. |
| Number of Updates Per Duration | Specifies how frequently users will be updated with content within the specified delivery time window. Note that this option is for push delivery only. |
| Days of Week | Specifies the days of the week when content is to be delivered. If you wish to specify days of the week, select **Only**, and check the days of the week when content is to be delivered. Selecting **Any** means there are no limitations for daily delivery during the week. |
| Days of Month | Specifies the days of the month when content is to be delivered. If you wish to specify the days of the month, select **Only** and check the days of the month when content is to be delivered. Selecting **Any** means there are no limitations for daily delivery during the month. **LAST** means the last day of the month only. |

**Note:** If you want to limit content delivery to days of the week or days of the month, choose the appropriate option. If you choose to use both options together to limit content delivery, both conditions must be met. For example, selecting day of the week as M or Mondays, and days of the month as 3, will restrict content delivery only to all Mondays that are also the 3rd day of the month.

- **Push delivery rules** -- **Enable Push Delivery** selected: a set of controls automatically delivers content to subscribed users on a time-based schedule. Push delivery rules describe the daily time period during which content pushes are allowed (see Table 2–8), starting at the time specified and lasting for the duration specified. Specify the time zone, the start time in hours and minutes for push content delivery, the total duration in hours and minutes for the delivery of content, the number of content updates allowed per delivery time period, and the days of the week or days of the month in which push delivery is allowed.

---

**Note:** If you want to limit content delivery to days of the week or days of the month, choose the appropriate option. If you choose to use both options together to limit content delivery, both conditions must be met. For example, selecting day of the week as M or Mondays, and days of the month as 3, will restrict content delivery only to all Mondays that are also the 3rd day of the month.

---

#### 2.5.4.2 Edit the Contract Business Terms

To edit the business terms of a contract, click the **Business Terms** tab. An **Edit Contract Business Terms** page is displayed. On this page, the terms and conditions that define the agreement that users must accept upon subscribing to the offer are defined. You can edit the name and description for your terms and conditions, and edit the agreement details associated with the contract. When you have finished making edits, click **OK** to save your changes, which returns you to the **Offer Management Contracts** page.

#### 2.5.4.3 Delete a Contract

To delete a contract, click its radio button in the **Select** column, then click **Delete**. A warning message is displayed indicating that deleting a contract will also change the access policies for the associated offers. Are you sure you want to proceed? Click **Delete** to confirm the deletion of the contract.

#### 2.5.4.4 Create a Contract

To create a contract, click **Create**. A **Create Contract General** page is displayed. On this page, you can create the contract by filling in the details, which include:

- **General properties** -- contract name, contract description, and requires confirmation. Requires confirmation means whether or not confirmation of receipt of the content package is required before the next content package can be delivered to the user. See Section 2.9.2 for more information.
- **Expiration policy** -- expiration information for subscriptions based on this contract (see Table 2–6), such as the start and stop dates, including the time zone, and maximum number of deliveries for the subscription, and the expiration priority (see Table 2–7): time-based, quantity-based, first, or last.
- **Pull delivery rules** -- **Enable Pull Delivery** selected: a set of controls delivers content to subscribed users on demand. Describes the daily time period during which content pulls are allowed (see Table 2–8), starting at the time specified and lasting for the duration specified. Specify the time zone, the start time in

hours and minutes for pull content delivery, the valid duration in hours and minutes for the delivery of content, and the days of the week or days of the month in which pull delivery is allowed.

> **Note:** If you want to limit content delivery to days of the week or days of the month, choose the appropriate option. If you choose to use both options together to limit content delivery, both conditions must be met. For example, selecting day of the week as M or Mondays, and days of the month as 3, will restrict content delivery only to all Mondays that are also the 3rd day of the month.

- **Push delivery rules** -- **Enable Push Delivery** selected: a set of controls automatically delivers content to subscribed users on a time-based schedule. Push delivery rules describe the daily time period during which content pushes are allowed (see Table 2–8), starting at the time specified and lasting for the duration specified. Specify the time zone, the start time in hours and minutes for push content delivery, the total duration in hours and minutes for the delivery of content, the number of content updates allowed per duration period, and the days of the week or days of the month in which push delivery is allowed.

> **Note:** If you want to limit content delivery to days of the week or days of the month, choose the appropriate option. If you choose to use both options together to limit content delivery, both conditions must be met. For example, selecting day of the week as M or Mondays, and days of the month as 3, will restrict content delivery only to all Mondays that are also the 3rd day of the month.

#### 2.5.4.5 Create the Contract Business Terms

To create the business terms of a contract, click the **Business Terms** tab. A **Create Contract Business Terms** page is displayed. On this page, the business terms that define the agreement that users must accept upon subscribing to the offer can be defined. You can enter the name and description for your business terms, and enter the agreement details associated with the contract. When you have finished entering the details of the business terms, click **OK** to save your changes, which returns you to the **Offer Management Contracts** page.

## 2.6 Managing Subscriptions

After users subscribe to an offer, a subscription is created. You can view the list of subscriptions on the **Subscription Management** page.

At the **Oracle Enterprise Manager Syndication Services Administration** page, click **Subscriptions**. The **Subscription Management** page is displayed. On this page, you can:

- View or Edit a Subscription (General Information)
- View or Edit Offer Information for This Subscription
- Terminate a Subscription
- Purge Expired or Terminated Subscriptions

### 2.6.1 View or Edit a Subscription (General Information)

To view or edit a subscription, at the **Subscription Management** page, click the subscription name in the **Subscription** column that you want to edit, or select the subscription by clicking its radio button in the **Select** column, and then click **Edit**. The **Edit Subscription General** page is displayed. On this page, general subscription information is categorized into the following groups:

- **General Information** -- subscribed user name, start date of the subscription, package sequence state, and the state of the subscription: Active, Suspended, or Terminated. The state of the subscription is the only subscription information you can change. To temporarily end a subscription, you can specify a state of Suspended. If a subscription is suspended, its user will not be allowed pull or push content deliveries until the subscription is reactivated. A subscription whose state is Suspended can be changed back to a state of Active. However, if you change the state to Terminated, you cannot change the state of the subscription back to Active. When the subscription state is set to Terminated, the subscription is permanently terminated and it can be purged only by clicking **Purge Expired and Terminated** on the **Subscription Management** page.
- **Confirmation** -- whether or not confirmation of receipt of the content package for this subscription is required before the next content package can be delivered to the user.
- **Expiration Policy** -- whether or not the expiration of the subscription is time-based or quantity-based or a combination of both (see Table 2–7 for a description of these expiration priority categories), the date and time the

subscription expires, and the number of deliveries remaining for the subscription. See Table 2–6 for descriptions of these expiration policy categories.

- **Delivery Rules** -- how content is to be delivered as either a pull or a push, or both. Pull delivery indicates users receive content on demand, while push delivery indicates content is automatically delivered to subscribed users based on a time-based schedule. For both pull and push delivery rules, the time zone is indicated along with a start time (hour and minutes). The total duration in hours and minutes allowed for the content to be delivered is specified. The days of the week or of the month in which the delivery is to occur are specified. Finally, the date and time when the last delivery was completed is also specified. For push delivery only, the number of updates allowed per day is also indicated. See Table 2–8 for a description of these delivery rule categories.

### 2.6.2 View or Edit Offer Information for This Subscription

To view or edit offer information for this subscription, at the **Subscription Management** page, click the **Offer** tab. The **Edit Subscription Offer** page is displayed. On this page, offer information about this subscription is categorized into the following groups. Note that you cannot edit any of these fields.

- **General information** -- the offer description.
- **Content Usage Properties** -- product name, the name of the holder of the intellectual property rights, the content status of the intellectual property rights (see Table 2–4); and content usage patterns (see Table 2–5).
- **Content Provider information** -- name and the resource name.

### 2.6.3 Terminate a Subscription

To terminate a subscription, at the **Subscription Management** page, click the subscription name in the **Subscription** column that you want to terminate, or select the subscription by clicking its radio button in the **Select** column, and then click **Terminate**. The subscription's state is changed to Terminated. Note that you cannot change the state of a subscription to Active once it is set to Terminated. You can purge only the subscription whose state is Terminated by clicking **Purge Expired and Terminated** on the **Subscription Management** page.

### 2.6.4 Purge Expired or Terminated Subscriptions

To purge expired and terminated subscriptions, at the **Subscription Management** page, click **Purge Expired and Terminated**. All expired and terminated subscriptions are removed form the list of subscriptions.

## 2.7 Reviewing and Purging an Access Log

With access logging enabled, you can review an access log that contains entries for each access operation performed by users who use Syndication Services.

At the **Oracle Enterprise Manager Syndication Services Administration** page, click **Access Logs**. The **Access Logs** page is displayed. On this page, a search form lets the administrator select a subset of entry records filtered by various criteria. Using the basic or the advanced search form (note the expanding link), the administrator can search access entries in the log by user, type of event (access or admin), subscription ID, and date-time range. The result of the search will always be limited to a maximum of 200 entries. Clicking **GO** with no values specified in the search form will result in a full list of system events. Select an entry by clicking its radio button in the **Select** column, and then click **View**. A **View Log** page is displayed. This page contains a detailed entry record for a particular user's access to Syndication Services. Reviewing the details of this entry record can be useful for troubleshooting content delivery problems.

## 2.8 Monitoring Performance and Tuning

The Syndication Services servlet engine within `syndserver.ear`, and the associated JDBC connection pools to the Oracle Application Server Metadata Repository, can all be monitored using Oracle Enterprise Manager and other standard database monitoring and tuning utilities.

In an OC4J standalone environment, performance information is typically available at

```
http://<oc4j-host-name>:<port-number>/dmsoc4j/Spy
```

## 2.9 Performing Advanced Administration Tasks

This section describes some advanced administration tasks. Section 2.9.1 describes how Syndication Services administration features map to an ICE element and attribute for contracts and business terms, offers, and Syndication Services properties. Section 2.9.2 describes how confirmations work and Section 2.9.3 describes the negotiation process.

### 2.9.1 How Syndication Services Administration Features Map to ICE in Terms of Modeling

Section 2.9.1.1, Section 2.9.1.2, and Section 2.9.1.3 describe how Syndication Services administration features for contracts and business terms, offers, and Syndication Services properties are modeled to the ICE 1.1 element and attributes.

#### 2.9.1.1 Contracts and Business Terms

Table 2–9 shows the mapping of contract properties by administrative field of the Syndication Services client API to the ICE element and attribute.

*Table 2–9 Contract Properties Mapping*

| Administration Field | Syndication Services Client API | ICE Element and Attribute | Notes |
|---|---|---|---|
| Name | NA | NA | Only used by administrator to identify the contract. |
| Description | NA | NA | Only used by administrator to describe the contract. |
| Requires confirmation | NA | NA | Once selected, all subscriptions based on this contract will require confirmation for any package delivery. See Section 2.9.2 for more information. |
| Expiration priority | Offer.getExpirationPriority() | expiration-priority@ice-offer | |
| Start date | DeliveryPolicy.getStartDate() | start-date@ice-delivery-policy | |
| Stop date | DeliveryPolicy.getStopDate() | stop-date@ice-delivery-policy | |
| Quantity | Offer.getQuantity() | quantity@ice-offer | |
| Enable pull delivery | NA | NA | Once selected, this will add a pull delivery rule in the corresponding ICE-offer. |
| Pull start time | DeliveryRule.getStartTime() | start-time@ice-delivery-rule | DeliveryRule mode set to be PULL to differentiate from PUSH DeliveryRule. |

***Table 2–9 Contract Properties Mapping (Cont.)***

| Administration Field | Syndication Services Client API | ICE Element and Attribute | Notes |
|---|---|---|---|
| Pull duration | DeliveryRule.getDuration() | duration@ice-delivery-rule | DeliveryRule mode set to be PULL to differentiate from PUSH DeliveryRule. |
| Pull days of week | DeliveryRule.getWeekDay () | weekday@ice-delivery-rule | DeliveryRule mode set to be PULL to differentiate from PUSH DeliveryRule. |
| Pull days of month | DeliveryRule.getMonthDay() | monthday@ice-delivery-rule | DeliveryRule mode set to be PULL to differentiate from PUSH DeliveryRule. |
| Enable push delivery | NA | NA | Once selected, this will add a push delivery rule in the corresponding ICE-offer. |
| Push start time | DeliveryRule.getStartTime() | start-time@ice-delivery-rule | DeliveryRule mode set to be PUSH to differentiate from PULL DeliveryRule. |
| Push duration | DeliveryRule.getDuration() | duration@ice-delivery-rule | DeliveryRule mode set to be PUSH to differentiate from PULL DeliveryRule. |
| Push number of updates per duration | DeliveryRule.getMinNumUpdates() | min-num-updates@ice-delivery-rule | DeliveryRule mode set to be PUSH to differentiate from PULL DeliveryRule. |
| Push days of week | DeliveryRule.getWeekDay() | weekday@ice-delivery-rule | DeliveryRule mode set to be PUSH to differentiate from PULL DeliveryRule. |
| Push days of month | DeliveryRule.getMonthDay() | monthday@ice-delivery-rule | DeliveryRule mode set to be PUSH to differentiate from PULL DeliveryRule. |

Table 2–10 shows the mapping of business terms properties by administrative field of the Syndication Services client API to the ICE element and attribute.

***Table 2–10 Business Terms Properties Mapping***

| Administration Field | Syndication Services Client API | ICE Element and Attribute | Notes |
|---|---|---|---|
| Name | BusinessTerm.getName() | name@ice-business-term | |
| Business terms | BusinessTerm.getText() | text@ice-business-term | |
| NA | NA | type@ice-business-term | Always set to "licensing" |

### 2.9.1.2 Offer Properties

Table 2–11 shows the mapping of offer properties by administrative field of the Syndication Services client API to the ICE element and attribute.

***Table 2–11 Offer Properties Mapping***

| Administration Field | Syndication Services Client API | ICE Element and Attribute | Notes |
|---|---|---|---|
| Name | NA | NA | Used to identify the offer. |
| Description | Offer.getDescription() | description@ice-offer | |
| Offer state | NA | NA | Used by the administrator to control the state of the offer. Only offers whose state is active are visible to users when requesting a catalog. |
| Product name | Offer.getProductName() | product-name@ice-offer | |
| Rights holder | Offer.getRightsHolder() | rights-holder@ice-offer | |
| Intellectual property status | Offer.getIPStatus() | ip-status@ice-offer | |
| Atomic use | Offer.isAtomic () | atomic-use@ice-offer | |
| Editable | Offer.isEditable() | editable@ice-offer | |
| Show credit | Offer.hasShowCredit() | show-credit@ice-offer | |
| Usage required | Offer.hasUsageRequired() | usage-required@ice-offer | |
| Content provider name | NA | NA | Used to identify where this offer comes from. |
| Resource name | NA | NA | Used to identify the resource associated with this offer from the content provider side. |

### 2.9.1.3 Syndication Services Properties

Table 2–12 shows the mapping of Syndication Services properties by administrative field of the Syndication Services client API to the ICE element and attribute.

***Table 2–12 Syndication Services Properties Mapping***

| Administration Field | Syndication Services Client API | ICE Element and Attribute |
|---|---|---|
| Instance ID | Sender.getSenderID() | sender-id@ice-sender |
| Instance name | Sender.getName() | name@ice-sender |
| User agent name | Header.getUserAgent() | ice-user-agent/ice-header |

## 2.9.2 Description of Confirmations

Syndication Services delivers content in the form of packages and can request its users to confirm the correct delivery of a content package before the next one can be sent. As the server will send only the content updates since the last request, no new updates will be sent before the server receives a confirmation that everything sent so far has been correctly received and processed.

This implies that when the confirmation flag is set for a contract, all subscriptions depending on that contract will require users to send an explicit confirmation before they can receive new content. This behavior applies to both pull and push delivery rules. For example, if users try to pull content updates before sending a confirmation, Syndication Services will respond with the exception, `ICE-602: Excessive confirmations outstanding`. This is the default behavior.

The following two properties can be used to set bounds for not receiving content delivery confirmations from users, or for allowing users not to confirm or reject too many content deliveries between content delivery confirmations:

- `oracle.syndicate.server.delivery.max_outstanding_confirms`

  Maximum number of outstanding confirmations -- for a given subscription that requires confirmation from the user (subscriber) of the delivery of a content package, this property defines the maximum number of content packages that will be delivered to the user following lack of confirmation. For example, setting a value of 5 means only 5 content packages will be delivered for this subscription following an unconfirmed delivery; the sixth and following deliveries will be suspended until a confirmation is received. See Section 2.9.5 for more information about this property.

- `oracle.syndicate.server.delivery.max_unconfirmed_packages`

  Maximum number of unconfirmed packages -- for a given subscription that requires confirmation from the user (subscriber) of the delivery of a content package, this property defines the maximum number of unconfirmed or rejected content packages that are allowed between two delivery confirmations.

For example, setting a value of 2 means that the user cannot leave a package receipt unconfirmed or reject more than 2 content deliveries between any 2 content delivery confirmations. This parameter can be used to prevent users from receiving more content package deliveries than their contract stipulates. See Section 2.9.5 for more information about this property.

### 2.9.3 Negotiation

Syndication Services lets users customize the offer at subscription time. The supported customizations are limited to the following modifications:

- Users can select the set of delivery rules they would like to use for their subscriptions among those supplied in the offers. For example, if a user is presented an offer that features both the pull and push delivery rules, the user can decide at subscription time to subscribe only to the pull delivery rule, therefore omitting the push delivery rule from the subscription request.
- Users can customize the push delivery rules by supplying the push URL (which can be based on `http`, `https`, `ftp`, or `mailto`), and the content delivery option (being ICE or non-ICE deliveries). If the URL provided requires authentication, users can provide the necessary credentials. See Section 3.8.3 for more information.

If the user modifies the offer supplied in the subscription request in any other form, Syndication Services will reject the request and reply with an exception, `ICE-441: Counter-proposal`.

### 2.9.4 Scheduler Configuration

Syndication Services features a scheduler component to manage time-based content pushes. The scheduler component is part of the Syndication Services J2EE application and is deployed with the Syndication Services installation. This section describes the operations of the scheduler and the properties that can be used to tune it.

#### 2.9.4.1 Assign New Scheduler Rules

Upon the creation of a new subscription featuring a push delivery rule, Syndication Services stores a new scheduler rule in the Oracle Application Server Metadata Repository. A scheduler rule is used to compute the frequency of updates and their specific push date and time. The scheduler component will periodically check for new scheduler rules in the metadata repository. If a new scheduler rule is found, the scheduler will take ownership of that rule and start handling its tasks from that time on.

The `monitor frequency` system parameter, available on the System Properties page in Syndication Services administration, controls how frequently the scheduler component will check for new rules. The default monitor frequency value is one minute, which means that, at most, the push actions associated with a new subscription will be taken care of one minute after subscription creation. You can increase the monitor frequency if you want a more prompt handling of newly created subscriptions; however, this will result in more frequent round trips to the metadata repository, thus having an impact on the overall system performance.

#### 2.9.4.2 Handle Scheduler Rules

Once a rule is owned by a scheduler instance, the scheduler component will compute date and time of the next push tasks and schedule a timer for them. The `timer pool size` system parameter, available in the **System Properties** page in Syndication Services administration, controls how many timers are concurrently active in each scheduler instance. The default value is 16, meaning that, at any point in time, the scheduler will have 16 active timers -- one for each of the next 16 push tasks across all the scheduler rules assigned to that instance. You can increase the size of the timer pool if you plan to have a high number of tasks close in time.

#### 2.9.4.3 Recover Scheduler After System Shutdown or Restart

When Syndication Services is shut down (in a default installation that means shutting down the OC4J_Portal instance), the scheduler component will also be shut down. Scheduled push deliveries will not be executed while Syndication Services is shut down.

Upon restart, each Syndication Services scheduler instance will recover the scheduler rules assigned to it and resume the content push deliveries. If push actions were scheduled during the down time and therefore not executed, they will be performed in fast sequence upon restart before resuming normal operations.

If one OC4J instance goes down due to a software failure, it will be automatically recovered by Oracle Process Manager and Notification (OPMN) Server. Upon recovery, the scheduler rules associated with that instance will be recovered following the behavior just described.

#### 2.9.4.4 Use of Syndication Services Scheduler in an Oracle Application Server Cluster

In a clustered environment where Syndication Services is deployed to more than one OC4J instance, all Syndication Services scheduler components will periodically check for new scheduler rules. However, once assigned, each scheduler rule is owned by one OC4J instance only. The assignment process distributes the scheduler rules uniformly across all available OC4J instances, thus achieving load balancing. Each OC4J instance will therefore own a set of scheduler rules; this means all OC4J instances must be up and running in order to handle all the scheduler rules.

The Syndication Services scheduler instance uses OC4J_ID to identify the ownership of rules to be scheduled. Whenever islands or OC4J instances where Syndication Services is deployed are removed, rules marked for those islands or OC4J instances must be released and taken over by other Syndication Services scheduler instances.

Syndication Services uses the distributed configuration management (DCM) business rule framework to handle these cases. The DCM business rule framework allows components, such as Syndication Services, to register its plug-in and invoke corresponding methods when related events happen. The syntax to register the plug-in is:

```
$<MIDTIER_ORACLE_HOME>/dcm/bin/dcmctl registerplugin -f $<MIDTIER_ORACLE_
HOME>/syndication/config/bizruleReg.xml
```

Once the plug-in is registered, Syndication Services will listen for any OC4J instance removal and island removal events. Whenever these events occur, the plug-in cleans up corresponding ownership flags and other Syndication Services scheduler instances subsequently pick up these rules.

### 2.9.5 Syndication Services Advanced Properties

This section describes several Syndication Services advanced properties. These advanced properties can be set only by following the procedure described in Section 2.9.5.1. These advanced properties include:

- `oracle.syndicate.transport.ftp.act_connect_mode` -- an FTP transport property that lets you select active or passive FTPmode. Possible values are `true` or `false`. The default value is `false` or passive FTPmode.
- `oracle.syndicate.server.delivery.max_outstanding_confirms` -- a property that defines the maximum number of content packages that will be delivered to the user (subscriber) following an unconfirmed delivery; further

deliveries are suspended until the delivery is confirmed. Possible values are any positive integer. The default value is 1.

- `oracle.syndicate.server.delivery.max_unconfirmed_packages` -- a property that defines the maximum number of unconfirmed or rejected content packages that are allowed between two delivery confirmations before deliveries are suspended. Possible values are any positive integer. The default value is 1.

#### 2.9.5.1 How to Set Advanced Properties for Syndication Services

Advanced properties are used to fine tune the functions of Syndication Services. Customization of such parameters is expected to be uncommon, and therefore these properties cannot be set using the Oracle Application Server Syndication Services administration pages. These advanced properties must be set directly in Oracle Application Server Metadata Repository.

The procedure to be followed to set these advanced properties involves opening a SQL*Plus session in the Oracle Application Server Metadata Repository and executing a SQL statement to set the preferred property value.

In a UNIX environment, you can open a SQL*Plus session with the following set of commands:

```
cd <ORACLE_HOME>
setenv ORACLE_HOME <ORACLE_HOME>
setenv ORACLE_SID  <ORACLE_SID>
bin/sqlplus "sys/<sys_password> as sysdba"
```

In the SQL*Plus session, use the following command to set the session schema to the Syndication Services schema:

```
alter session set current_schema=dsgateway;
```

The PROPERTIES table contains the property name, value, and description of the properties used by Syndication Services. The current value for a property can be verified by performing a `SELECT` statement on the table.

```
SELECT propname, propvalue
FROM PROPERTIES;
```

Properties values can be updated using the following syntax:

```
update PROPERTIES set PROPVALUE = 'pvalue' where PROPNAME = 'pname';
```

New properties can inserted using the following syntax:

```
insert into PROPERTIES (PROPNAME, PROPVALUE, DESCRIPTION) VALUES ('pname',
'pvalue', 'pdesc');
```

# 3 Client Application Development

This chapter describes client application development.

Syndication Services provides a Java client library, which can be used to integrate Syndication Services in Java-based Java 2 Platform, Standard Edition (J2SE), or Java 2 Platform, Enterprise Edition (J2EE) applications. The Syndication Services client library lets users request the catalog of offers granted to them, subscribe to one or more of these offers, and receive the associated content. The following sections will describe the code sequence of a sample Syndication Services client application performing the following:

- Opening a Syndication Services Connection
- Acquiring a Catalog of Offers
- Subscribing to an Offer
- Delivering Content
- Verifying Subscription Expiration Status
- Canceling a Subscription
- Verifying Activity Associated with a Subscription

A set of demonstration files, including the full source file of this example, can be downloaded from the Oracle Technology Network (OTN) Web site `http://otn.oracle.com/tech/java/oc4j/demos/`

A complete listing of the `SampleSyndicationClient.java` program, parts of which are described in this chapter, can be found in Section C.1. The API documentation that describes how to use the Syndication Services client API can be found on the Oracle Application Server 10*g* Documentation Library as Oracle Application Server Syndication Services API Reference (Javadoc) under Oracle Application Server Portal. The client library, `syndclient.jar`, is located in

*$ORACLE_HOME*/syndication/lib on UNIX and in <*ORACLE_HOME*>\syndication\lib on Windows.

The Syndication Services client library is modeled after the Information Content Exchange (ICE) 1.1 standard. The library is indeed acting as an ICE client in its information exchanges with Syndication Services, which is an ICE 1.1-compliant server. The ICE standard is a content syndication standard defining data structures and information exchanges for establishing syndication relationships. While implementing a generic and interoperable ICE client, the following documentation on the Syndication Services client library will focus on the subset of operations, data structures, and attributes that are relevant to its usage when accessing the Syndication Services server. More detailed information on how Syndication Services maps to ICE can be found in Section 2.9.1.1, Section 2.9.1.2, and Section 2.9.1.3. The ICE specification 1.1 can be found at the following URL
http://www.icestandard.org/

## 3.1 Opening a Syndication Services Connection

When building applications, which interact with Syndication Services, client code should first acquire a SyndicateConnection instance. Example 3–1 illustrates how SyndicateConnection instance can be acquired.

***Example 3–1 Acquiring a SyndicateConnection Instance***

```
// Acquire a SyndicateConnectionFactory,
// optionally set the connection parameters,
// such as proxy info, timeout, and credentials.
SyndicateConnectionFactory scf = SyndicateConnectionFactory.getInstance();
// scf.setTimeout(1000);
// scf.setProxy("myproxyhost", iMyProxtPort);
// Create a default XML state handler storing subscription state in a file.
SyndicateClientStateHandler scsh =
 scf.getDefaultSyndicateClientStateHandler("synd-client.xml");
SyndicateConnection sc =
 scf.createSyndicateConnection("http://myias/syndserver/server",  // server url
                                    "myusername",  // username
                                    "mypassword",  // password
                                    scsh);
```

A SyndicateConnection instance is acquired through the SyndicateConnectionFactory object. The latter contains APIs to set connection properties before their creations. Such properties include: timeout, credentials to be used in case of an HTTPS connection, and the HTTP proxy to be used. Opening a

SyndicateConnection instance implies establishing an HTTP connection with Syndication Services, and authenticating as a valid user. In a default Oracle Application Server installation, the Syndication Services server is deployed in the OC4J_Portal instance and the relative path of its URL is /syndserver/server (for example, http://myias:7777/syndserver/server). In order to successfully establish a connection to Syndication Services, the supplied user name and password needs to match the credentials of a valid Syndication Services user. See Section 2.3 for more information about the management of Syndication Services users.

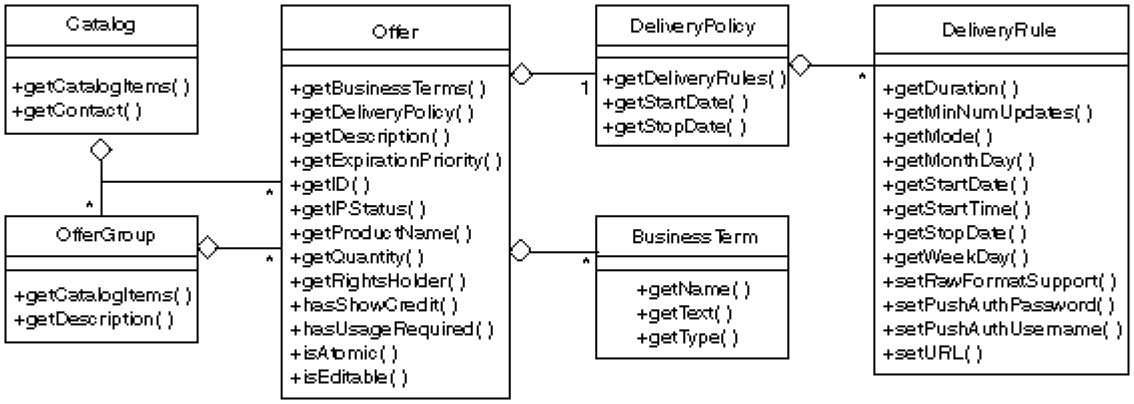
Syndication Services clients are generally *stateful*. In fact, they need to keep track of the set of subscriptions they created and, when receiving incremental content updates, the state of each of these subscriptions. An instance of SyndicateClientStateHandler can be optionally supplied when nesting a SyndicateConnection instance: SyndicateClientStateHandler exposes a set of interfaces called by the SyndicateConnection instance whenever a client state modification is required, such as subscription creation and updates. The default SyndicateClientStateHandler instance will store subscription state information into an XML file. Custom implementations of the SyndicateClientStateHandler interface, which will, for example, store the subscriptions state in a database, can also be supplied.

SyndicateConnection objects are not synchronized; programmers, who make calls are therefore expected to synchronize their access or create a pool of those connections if building a multithreaded application.

## 3.2 Acquiring a Catalog of Offers

**Catalogs** are a set of offers available for subscriptions. The catalog structure is illustrated in Figure 3–1 as a Unified Modeling Language (UML) diagram.

***Figure 3–1 Structure of a Catalog and Its Main Properties***



A catalog is a set of offers optionally grouped in offer groups. An offer represents a content unit, for example a directory in a repository, to which users can subscribe in order to receive content updates. Offers provide accessors for content usage properties, usage licenses, if any, and finally the modes and times in which the offer content will be delivered if a subscription is established. Example 3–2 shows how to acquire a catalog and browse through its offers. In this example, the catalog is grouped by content provider.

***Example 3–2 Acquiring a Catalog and Browsing Through Its Offers***

```
Catalog cat = sc.getCatalog();
Offer   off = getFirstOffer(cat);
}

  private Offer getFirstOffer(Catalog cat)
    throws SyndicateException
  {
    // Iterates through the catalog and returns the
    // first offer encountered.
    Offer   off = null;
    Iterator it = cat.getCatalogItems();
    while (it.hasNext() && (off == null)) {

      CatalogItem item = (CatalogItem)it.next();
      if (item.getCatalogItemType() == CatalogItem.OFFER) {
        off = (Offer) item;
      }
      else {
          // You got an offer group.
          off = getFirstOffer((OfferGroup) item);
        }
      }

      // Print a few offer details.
      System.out.println("offer: "+off.getID()+" - "+off.getDescription());
      DeliveryPolicy dp = off.getDeliveryPolicy();
      if (dp.getStartDate() != null) {
        System.out.println("\t start date: "+dp.getStartDate());
      }
      if (dp.getStopDate() != null) {
        System.out.println("\t  stop date: "+dp.getStopDate());
      }
      Iterator dlrs = dp.getDeliveryRules();
      while (dlrs.hasNext()) {
        DeliveryRule dlr = (DeliveryRule) dlrs.next();
        System.out.println("\t  dlr mode: "+dlr.getMode());
      }
      return off;
    }
  private Offer getFirstOffer(OfferGroup offgrp)
      throws SyndicateException
    {
      Offer   off = null;
      Iterator it = offgrp.getCatalogItems();
      while (it.hasNext() && (off == null)) {

        CatalogItem item = (CatalogItem)it.next();
        if (item.getCatalogItemType() == CatalogItem.OFFER) {

          off = (Offer) item;
        }
        else {
          // You got an offer group.
          off = getFirstOffer((OfferGroup) item);
        }
      }
      return off;
    }
```

## 3.3 Subscribing to an Offer

Example 3–3 shows how users can subscribe to an offer. Subscribing to an offer is achieved by customizing the supplied offer according to the user needs and calling the `SyndicateConnection.subscribe` method. A typical customization would include setting the destination address and the content format for pushed content packages. For more information regarding the offer customization allowed to users at subscription time, see Section 2.9.3.

***Example 3–3 Subscribing to an Offer***

```
// If the offer contains a push delivery rule,
// you can use the offer APIs to set the destination URL.
DeliveryPolicy dp = off.getDeliveryPolicy();
Iterator   itdlrs = dp.getDeliveryRules();
while (itdlrs.hasNext()) {

  DeliveryRule dlr = (DeliveryRule) itdlrs.next();
  if (DeliveryRule.DELIVERY_RULE_MODE_PUSH.equals(dlr.getMode())) {

    dlr.setURL("http://mysyndicationclient.com/syndclient/listener");
    // Optionally set user name/password.
    // dlr.setPushAuthUsername("me");
    // dlr.setPushAuthUsername("pwd");
    // If raw content is requested (for example, a mailto or
    // ftp url has been supplied), set the raw content flag.
    // dlr.setRawFormatSupport(true);
  }
}

Subscription sbt = sc.subscribe(off);
System.out.println("subscription: "+sbt.getSubscriptionID());
```

## 3.4 Delivering Content

Once a subscription to an offer is established, content can be delivered according to the time periods defined in the offer delivery rules. If the subscription contains push delivery rules, Syndication Services will schedule automatic push updates according to the delivery rule frequency. Refer to the API documentation, which can be found on the Oracle Application Server 10*g* Documentation Library as Oracle Application Server Syndication Services API Reference (Javadoc) under Oracle Application Server Portal, to see how to configure the supplied

SyndicateClientServlet servlet to build an HTTP-based listener, which, upon receiving a push content package, will extract the received content and store it in the local file system. Using the supplied Java APIs, users can use the SyndicateClientServlet servlet to perform custom handling of the received content.

If the subscription contains pull delivery rules, users can request a content update as shown in Example 3–4.

***Example 3–4 Requesting a Content Update***

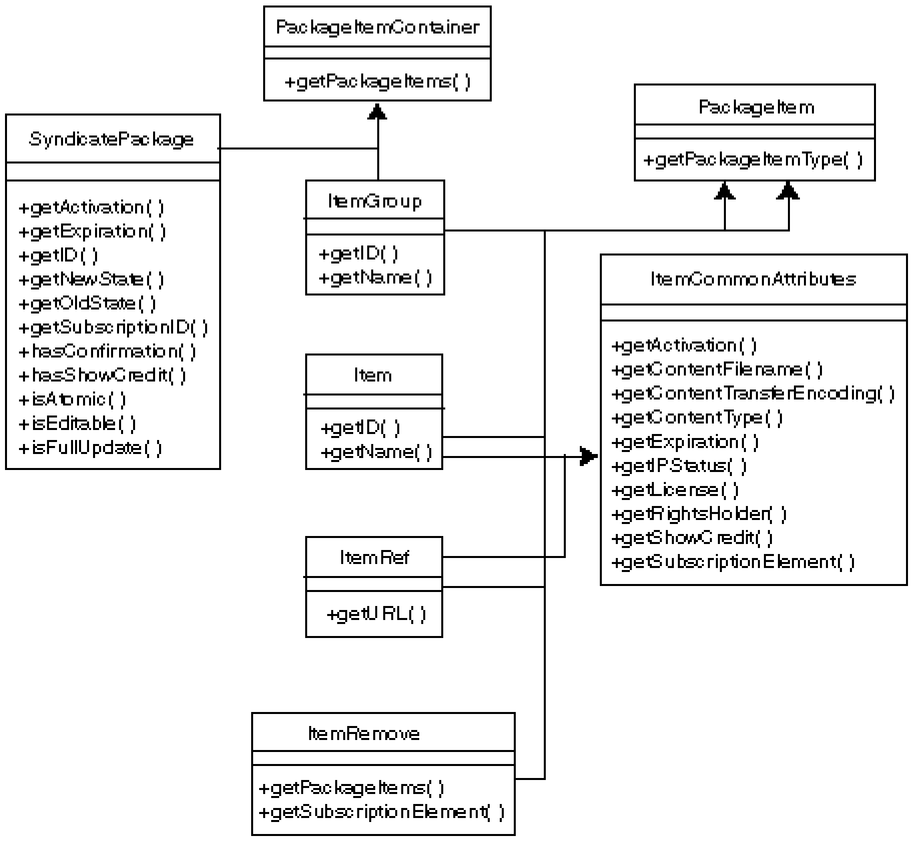
```
// Use a FileSAXPackageHandler instance so that
// syndicate content will be stored in
// the local file system /tmp directory.
FileSAXPackageHandler fsph = FileSAXPackageHandler.getInstance("/tmp/");

// This will get content incremented since the last update.
// The current subscriber state will be fetched from
// the SyndicateClientStateHandler object used by this connection.
// To request a full update of the content versus an incremental
// one, use the following syntax:
// sc.getPackage(sbt.getSubscriptionID(),
//                SyndicateSubscription.STATE_ICE_INITIAL,
//                null,
//                fsph);
SyndicatePackage pkg = sc.getPackage(sbt.getSubscriptionID(), null, fsph);
```

The preceding code sample uses the supplied FileSAXPackageHandler instance to have the received content stored in a file system directory. A custom package handler can be developed using the Syndication Services client library by implementing the SAXPackageHandler interface. Refer to the API documentation, which can be found on the Oracle Application Server 10*g* Documentation Library as Oracle Application Server Syndication Services API Reference (Javadoc) under Oracle Application Server Portal, for more information.

The returned SyndicatePackage object can be used to iterate through the set of items received in this content update. Figure 3–2 shows a UML diagram of the structure of a content package and its main properties.

***Figure 3–2 Structure of a Content Package and Its Main Properties***



## 3.4.1 Package Confirmation Requests

Syndication Services can request its users to confirm the correct delivery of a content package before the next one can be sent. As the server will send only the content updates since the last request, no new updates will be sent before the server receives a confirmation that everything sent so far has been correctly received and processed.

For push delivery rules whose listener implementation is based on the supplied `SyndicateClientServlet` servlet, the servlet code will automatically confirm a pushed content package if the content package has been correctly processed by the specified `SAX Package Handler` interface. Also, if the push URL is SMTP- or

FTP-based, if no transport level error is detected during the content transfer, the user state is automatically updated and an implicit confirmation is assumed.

For pull delivery rules, if the received content package requires a confirmation, Syndication Services requires a confirmation to be received before additional deliveries (pull or push) can be performed. Example 3–5 shows how to verify if a confirmation is required and, if so, how to send such a confirmation.

***Example 3–5 Verifying If a Confirmation Is Required and Sending a Confirmation***

```
    // Check if the content package requires confirmation.
    // If so, confirm it.
    if (pkg.hasConfirmation()) {

     Response resp = _sc.confirm(pkg.getID());
      System.out.println("confirmed "+pkg.getID()+":
 "+resp.getCode().getPhrase());
    }
```

## 3.5 Verifying Subscription Expiration Status

Example 3–6 shows how the status of a subscription can be verified. The returned information can be used to determine the subscription settings and expiration criteria.

***Example 3–6 Verifying the Status of a Subscription***

```
    Status sts = sc.getStatus(sbt.getSubscriptionID());

    Subscription sbtNew = (Subscription) sts.getSubscriptions().next();
    System.out.println(" sbt "+sbtNew.getSubscriptionID()+":");

    int pri = sbtNew.getExpirationPriority();
System.out.println("     expiration priority: "+
 Subscription.EXPIRATION_PRIORITIES[pri]);
    System.out.println("     expiration date: "+sbtNew.getExpirationDate());
    System.out.println("     quantity
 remaining:"+sbtNew.getQuantityRemaining());
```

## 3.6 Canceling a Subscription

Users can decide to terminate a subscription before it expires. Example 3–7 illustrates how subscriptions can be terminated.

***Example 3–7 Terminating a Subscription***

```
Cancellation canc = sc.cancelSubscription(sbt.getSubscriptionID(),
                                                         "boring", "en_us");
System.out.println("cancelled: "+canc.getCancellationID());
```

## 3.7 Verifying Activity Associated with a Subscription

Users can request Syndication Services for a log of the activity associated with a given subscription. Optionally, a time frame can be supplied to limit the set of events returned. Example 3–8 illustrates how the activity of a subscription can be retrieved.

***Example 3–8 Retrieving the Activity of a Subscription***

```
    Events evt = sc.getEvents(null, null, sbt.getSubscriptionID());
    EventLog evtlog = evt.getEventLog();
    Iterator itEvts = evtlog.getEventItems();
    while (itEvts.hasNext()) {

      EventItem ei = (EventItem) itEvts.next();
      if (ei.getType() ==  EventItem.EVENT_TYPE_MSG) {

        EventMsg evtmsg = (EventMsg) ei;
        System.out.println(evtmsg.getRequestStart()+" "
                           +evtmsg.getRequest()+" "+
                           evtmsg.getResponseType());
      }
    }
  }
}
```

## 3.8 Client Library Reference Information

Section 3.8.1 through Section 3.8.5 describe interface attributes relevant to the Syndication Services server. For more information about these interfaces, see the

Oracle Application Server Syndication Services API Reference (Javadoc) that describes how to use the Syndication Services client API; this API documentation can be found on the Oracle Application Server 10*g* Documentation Library.

### 3.8.1 Offer Properties

This section describes the following offer attributes relevant to the Syndication Services server:

- **ID** -- the ID of the offer. It uniquely identifies an offer.
- **Description** -- a textual description of the offer and the content associated with it. The description helps a user understand if he is interested in subscribing to such an offer.
- **Product Name** -- a name that may be used to distinguish subscriptions and offers from other subscriptions or offers. Its intended use is to provide a readable short description of the offer such as, "Julia Child's Contemporary French Cooking Column".
- **Atomic Use** -- if true, all delivered content in the subscription must be used together, or not used at all. If false, then the user is permitted to use subsets of the data in any way desired (and as permitted by the licensing terms, of course).
- **Editable** -- if true, the user may edit or alter the content before using it. If false, the user is expected to use the content without any alteration.
- **Intellectual Property Status (IPStatus)** -- a string specifying the intellectual property-rights status of the content. ICE 1.1 defines the following specific string values:
  - PUBLIC-DOMAIN: the content has no licensing restrictions, whatsoever.
  - FREE-WITH-ACK: the content has no licensing restrictions beyond a requirement to display an acknowledgment of the content source.
  - SEE-LICENSE: the content has licensing restrictions as already agreed to in an existing licensing agreement. This is meant to convey the default case.
  - SEVERE-RESTRICTIONS: the content has licensing restrictions that are worthy of special attention.
  - CONFIDENTIAL: the content is confidential and must be protected specially.
- **Rights Holder** -- a string specifying the original source of the syndication rights.

- **Show Credit** -- if true, the subscriber is explicitly expected to acknowledge the source of the data.
- **Usage Required** -- whether or not the syndicator expects the subscriber to return usage data regarding the ultimate viewers of the syndicated content. Syndication Services does not define the format (or transport) of this usage data; however, this attribute lets a syndicator programmatically indicate the need for the data to the subscriber.

### 3.8.2 Business Terms

Business terms provide the means for additional content and parameters to be communicated and negotiated between the parties. They generally contain the license agreement that the syndicator wants the user to acknowledge as being acceptable before accepting a subscription and delivering the associated content.

### 3.8.3 Delivery Policy

The offer delivery policy describes the duration of a subscription. The **delivery policy start date** determines when the delivery of content starts. The expiration of the subscription is determined by a combination of attributes: the offer **expiration priority** attribute defines the criteria that will determine the subscription expiration. A subscription can expire after a certain number of content deliveries has been performed, or it can expired on a given date, or a combination of the preceding criteria. If the expiration priority value is `first`, then the subscription terminates when the first of the **quantity** or the **delivery policy stop date** is reached. If the expiration priority value is `last`, then the subscription terminates when both the **quantity** and the **delivery policy stop date** are reached. If the expiration priority value is `time`, then the subscription terminates when the **delivery policy stop date** is reached. If the expiration priority value is `quantity`, then the subscription terminates when the quantity is reached.

A delivery rule defines a period of time in which deliveries can be performed. Each delivery rule can be either a **push** (content is automatically delivered by the syndicator to the users) or **pull** (content delivery is requested by the user), can define which years, months, dates, and days of the week in which deliveries can be performed, a start and end time for the update period, and the count of the number of updates that can be performed.

The idea is to define a daily time period (defined by a start time and a duration) during which the deliveries happen. The start date, week day, month day, and stop day will define the day to which the daily time period will be applied. The most relevant delivery rule attributes are summarized as follows:

- **mode** -- a value of either `push` or `pull`. A push delivery means that the update is initiated by the syndicator. A pull delivery means that the update is initiated by the user.
- **Monthday** -- the delivery restricted to the given day of the month. This attribute is in the range 1 to 31, inclusive, or can be the special value `any` meaning no restrictions, or it can be the special value `last` meaning the last day of any month. Multiple values may be specified, separated by ' ' (a space).
- **Weekday** -- the delivery restricted to the given day of the week. This attribute must be in the range 1 to 7, inclusive, or be the special value `any`, meaning no restrictions. Multiple values may be specified, separated by ' ' (a space). In accordance with [ISO860] Section 5.2.3, the days are assigned as follows:
  - Monday is day number 1.
  - Tuesday is day number 2.
  - Wednesday is day number 3.
  - Thursday is day number 4.
  - Friday is day number 5.
  - Saturday is day number 6.
  - Sunday is day number 7.
- **Startdate** -- the beginning of the delivery rule time period. If earlier than the start date in the containing delivery policy, implementations must act *as if* this were equal to that start date. If no start date is supplied, the time period begins on the start date specified in the containing delivery policy. If there is no start date in the containing delivery policy, the time period begins immediately.
- **Stopdate** -- the end of the delivery rule time period. If later than the stop date in the containing delivery policy, implementations must act *as if* this were equal to that stop date. If no stop date is supplied, the delivery will end based on the stop date in the containing delivery policy, or, if the latter is not supplied, delivery never ends.
- **Starttime** -- the beginning time of a time period. If not defined in the subscription duration, a beginning time of 00:00:00 Coordinated Universal Time (UTC) will be applied. If the day under consideration is the first day, the time period begins at whichever is later: start time or the time portion of the start date.
- **Duration** -- the length of a daily time period. Duration will not exceed 24 hours. If the start time and duration are not supplied, the time period is supplied to

cover the whole day. If the start time is specified, and duration is not specified, then the semantics are *as if* a duration had been specified such that the time period will last from the start time until 24:00:00 UTC.

- **min-num-updates** -- the number of updates that the user is expected to receive during the time period identified by the start time and the duration. In Syndication Services, this parameter applies only to the push delivery rule.
- **url, pushAuthUsername and, pushAuthPassword** -- the address to which to send the update, if it is not to be sent to the normal ICE communication end point (see Table 3–1). In Syndication Services, the URL supplied may be either an HTTP or HTTPS protocol where a client listening to push packages is running. Other supported protocols include SMTP (for example, `mailto:myemail.com`) or FTP (for example, `ftp://myserver/mydir`). Optionally, if authentication is required, the user name and password of the user to be used can be supplied.

***Table 3–1 Push Delivery Options***

| Push URL Protocol | ICE Push (packageFormat=ICE_ PACKAGE_FORMAT) | Non-ICE Push (packageFormat=NON_ICE_ PACKAGE_FORMAT or NON_ICE_NO_ LOG_PACKAGE_FORMAT) |
|---|---|---|
| HTTP (for example, `http://user.com/syndclient/client`) | Syndication Services will perform an HTTP POST to the user HTTP end point containing the ICE XML package according to the ICE 1.1 specification. | Syndication Services will perform an HTTP POST to the user HTTP end point containing a MIME multipart message, where each part is a content package item. If the NON_ICE_PACKAGE_ FORMAT option is chosen, an additional part (the log) with a content package summary is also attached. |
| FTP (for example, `ftp://user.com/usr/homedir`) | Syndication Services will log in to the specified FTP site. It will change the current directory to `usr/homedir/`. It will then upload the ICE XML package as a single XML file whose name will be the `<package-id>.xml`. | Syndication Services will log in to the specified FTP site. It will change the current directory to `usr/homedir/`. It will then upload each content package item as a separate file retaining the item content-filename attribute and creating the necessary subdirectories. If the NON_ ICE_PACKAGE_FORMAT option is chosen, an additional file whose name is the `<package-id>.log` with a content package summary is also uploaded. |

***Table 3–1 Push Delivery Options (Cont.)***

| **Push URL Protocol** | **ICE Push (packageFormat=ICE_ PACKAGE_FORMAT)** | **Non-ICE Push (packageFormat=NON_ICE_ PACKAGE_FORMAT or NON_ICE_NO_ LOG_PACKAGE_FORMAT)** |
|---|---|---|
| SMTP (for example, `mailto:user@user.com`) | Syndication Services will send an e-mail message with a single part containing the ICE XML package. | Syndication Services will send an e-mail message with a mail attachment for each content package item. If the NON_ICE_ PACKAGE_FORMAT option is chosen, an additional text attachment (the log) with a content package summary is also attached. |

---

**Note:** `oracle.syndicate.transport.ftp.act_connect_mode`, an FTP property, controls FTP content pushes that are executed using active or passive FTP connection modes. See Section 2.9.5 for more information about this Syndication Services advanced property.

---

- **Raw format support** -- whether or not content updates are in the form of an XML payload (ICE XML package), a mail attachment (SMTP), or simply uploaded to the specified FTP server when an FTP URL is supplied.

  Syndication Services will send content updates to the specified address in the form of an XML payload (ICE XML package). It is up to the receiving client to understand the package structure and extract the content from it. Refer to the API documentation, which can be found on the Oracle Application Server 10*g* Documentation Library as Oracle Application Server Syndication Services API Reference (Javadoc) under Oracle Application Server Portal, to see how to configure the supplied `SyndicateClientServlet` servlet to build an HTTP-based listener, which, upon receiving a push content package, will extract the received content and store it in the local file system. Using the supplied Java APIs, users can use the `SyndicateClientServlet` servlet to perform custom handling of the received content.

  Syndication Services also offers the option for receiving the updated content without enclosing it into an ICE XML package. When setting this option, users will receive the updated files in the form of a mail attachment when using SMTP, or the updated content will be uploaded to the specified FTP server when an FTP URL is supplied.

### 3.8.4 Package Interfaces and Attributes

A `SyndicatePackage` interface describes a set of content operations: removals and additions. The remove operation is specified using the `ItemRemove` interface. The content additions contain the content that needs to be added or updated and is specified using the `Item` and `ItemRef` interfaces. The `ItemGroup` interface lets the syndicator associate the content specified using the `Item` and `ItemGroup` interfaces together. The `Item` and `ItemRef` interfaces distinguish themselves by the way they contain the content. The `Item` interface is used to contain content directly in the delivered content. The `ItemRef` interface is used to distribute an indirect reference to the actual content. The `SyndicatePackage` interface specifies an old state and a new state. Before the new state can be reached, all operations contained within a content package should be processed. If an operation cannot be performed successfully, all previously performed operations specified in the content package should be undone, so the user is not left in an inconsistent state. The `SyndicatePackage` interface exposes a set of accessor methods to manage its attributes. The most relevant attributes of the `SyndicatePackage` interface are summarized as follows:

- **activation** -- the date and time when the content contained in the content package may be *used*, as defined by the specific business relationship between the user and syndicator. If this same attribute is used on a specific item or group within the content package, the value specified there overrides this general specification within that part of the content package hierarchy. If this attribute is omitted, the content contained within the package is available for the user to deploy at his discretion.
- **atomic-use** -- a presentation constraint that either all content to be delivered must be used, or none of it is allowed to be used.
- **confirmation** -- whether or not the syndicator requires confirmation that the delivered content package was indeed delivered and processed successfully.
- **editable** -- whether or not the user may modify the information or must use it exactly as it is delivered.
- **expiration** -- the date and time after which the content contained in the package must not be used. If this same attribute is used on a specific item or group within the content package, the value specified there overrides this general specification within that part of the content package hierarchy.
- **fullupdate** -- whether or not the processing of the content package requires a full update of all content previously delivered in the subscription.

- **new-state** -- the state of the subscription after successfully processing the content package. This is an opaque value to the user, or something in which the user does not have direct access. The user can supply this state to the next `getPackage` request in order to receive the incremental update since then.
- **old-state** -- a string that specifies the state of the subscription before processing this content package.
- **package-id** -- the content package within the scope of a subscription. This is the value that will be used when confirming content packages, if required.
- **show-credit** -- whether or not the subscriber must prominently display attribution for the content source.
- **subscription-id** -- the subscription to which this content package belongs.

### 3.8.5 Attributes of the Item and ItemRef Interfaces

Attributes of the `Item` and `ItemRef` interfaces explicitly contain the content being distributed. The default content model for an attribute of the `Item` interface is character data. Binary data can be transmitted within an attribute of the `Item` interface by using a `base64` value. The attributes of the `ItemRef` interface manage the pointer to the content; its content is included in a content package delivery by reference. To learn how to access and process item content data, refer to the API documentation, which can be found on the Oracle Application Server 10*g* Documentation Library as Oracle Application Server Syndication Services API Reference (Javadoc) under Oracle Application Server Portal, for information about the `SAXPackageHandler` interface. The most relevant attributes of the `Item` and `ItemRef` interface are summarized as follows:

- **activation** -- the date and time when the content contained in the package may be *used*. If this same attribute is used on a specific item or group within the content package, the value specified there overrides this general specification within that part of the content package hierarchy.
- **expiration** -- the date and time when use of the content expires. If this same attribute is used on a specific item or group within the content package, the value specified there overrides this general specification within that part of the content package hierarchy.
- **content-filename** -- the relative destination for the item indicated by the syndicator. Paths stated in the `content-filename` attribute are represented using forward slashes as a path element delimiter. The subscription root is defined as a logical directory within which the user will place all file-based content received from the syndicator for that subscription; if the

`content-filename` attribute begins with a slash, the user *must* still treat the path as relative to the subscription root directory.

- **content-transfer-encoding** -- the encoding of the data contained in, or referenced by a `PackageItem` attribute. The only valid values are `x-native-xml` and `base64`. These values are derived from RFC-2045, sections 6.2 and 6.3. The `x-native-xml` value indicates that character entities may have been used to protect the special characters identified in Section 2.4 of the XML 1.0 W3C Recommendation. The `base64` value exhibits the same properties as that defined in Section 6.8 of RFC-2045. The `base64` value provides a means to distribute binary data, such as graphics formats, and other humanly unreadable data formats. Future versions of ICE may expand on this set of values.
- **content-type** -- the media type of the content, and should adhere to RFC-2045 and RFC-2046, including setting the `charset` parameter. The default value, `application/octet-stream`, indicates the opaque nature of the content being distributed.
- **ip-status** -- a string that specifies the intellectual property-rights status of the content.
- **item-id** -- the item uniquely identified within the scope of the containing content package.
- **rights-holder** -- the entity that holds the original syndication rights to the content in this element.
- **show-credit** -- whether or not the element's content requires crediting the source. This attribute *must* have a value of `true`, `yes`, `false`, or `no`. A value of `yes` or `true` indicates that use of this element's content requires crediting the source. A value of `no` or `false` means that source crediting is optional. The default value is `no`.
- **subscription-element** -- the content item persistently identified over the duration of the subscription. The only way to explicitly update or remove an individual content item is to use its subscription-element identifier in an operation in a subsequent content package. Alternatively, a full update will be required that will remove all of the content associated with the subscription.
- **name** -- the logical identifier for the content.

# 4 Content Connector Development

Content connectors (also known as Content Provider Adaptors or CPAdaptors) are software components used by Syndication Services to interact with external content repositories. Developers can create a content connector to expose resources of a repository as offers open for subscriptions.

Content connectors are developed using the Java programming language. To develop a content connector, you have to build a JavaBean, which implements the `CPAdaptor` interface in the `oracle.syndicate.server.cp` package. The following sections describe the code sequence of a sample content connector, which takes as input an RSS (Really Simple Syndication) feed (content source) and makes it a resource to which users can subscribe.

A set of demonstration files, including the full source file of this example, can be downloaded from the Oracle Technology Network (OTN) Web site `http://otn.oracle.com/`

A complete listing of the `RSSCPAdaptor.java` program, parts of which are described in this chapter, can be found in Section D.1. The API documentation that describes how to use the Syndication Services API for developing a content connector can be found on the Oracle Application Server 10*g* Documentation Library as Oracle Application Server Syndication Services API Reference (Javadoc) under Oracle Application Server Portal.

This chapter describes the following topics:

- Developing a Content Connector (CPAdaptor)
- Managing Content Connectors

## 4.1 Developing a Content Connector (CPAdaptor)

From a Java programming perspective, content connectors are JavaBean classes that implement the CPAdaptor interface in the `oracle.syndicate.server.cp` package. Content connector responsibilities include the following:

- Exposing a set of properties using the JavaBean *getter* and *setter* conventions. Typically, a connector will expose as properties those parameters that are required for its functions. For example, a connector that is an interface to an RDBMS system may decide to expose database connection parameters such as URL, user name, and password as connector properties. Only properties with Java primitive types can be edited in the Syndication Services administration pages during the registration of a content provider.
- Providing the list of resources to be made available to the Syndication Services administrator for offer creation.
- Building content packages associated with one of the content connector's resources. Content connectors can decide to build content packages incrementally, returning only the set of items that have changed since the last update, or as a full update, returning all the items associated with one of their resources, or both.

The `oracle.syndicate.server.cp.CPAdaptor` interface defines the set of APIs to be implemented by content connectors. Content connectors should be designed as *stateless java classes* whose only state information is determined by the values of their properties, and such values are determined by the administrator during the process of a content provider registration. The state is also immutable (cannot be changed) by the connector because the properties values can be set or modified through the administrator interfaces, and must not be altered by the connector during its operations. In summary, `CPAdaptor` classes must be carefully designed for a multithreaded environment, for example, multiple content package-building requests can be sent to the content provider concurrently.

The APIs are described by using a sample content connector, which reads an RSS feed and creates a resource for it. For more information regarding RSS and its specification, please refer to `http://backend.userland.com/rss`

The sample connector will create a content package item for every item defined in the RSS feed.

The sections that follow describe each stage of the content connector development process:

- Exposed Content Connector Properties

- Initialized Content Connector
- Exposed List of Resources
- Content Packages Built
- Incremental Updates
- Content Provider Closed
- Content Provider Event Push Support

### 4.1.1 Exposed Content Connector Properties

Content connectors can expose some of their settings as customizable properties. When building a generic content connector for a whole class of external content repository properties, exposing some of their settings can be used to customize a content provider instance to connect to one specific repository. In this respect, the content connector can be seen as a driver for a type of content management system, and its properties as the connection parameters to a specific repository instance.

The sample content connector described in Example 4–1 exposes only one property, which identifies the URL of the RSS feed to be exposed as a resource. The JavaBean `BeanInfo` classes can also be supplied to filter the set of properties exposed, and to add a property display name and description.

***Example 4–1 Exposing a Content Connector Property***

```
public class RSSCPAdaptor
  implements CPAdaptor, OSSExceptionConstants
{
  private static final String LAST_BUILD_DATE_FORMAT = "EEE, dd MMM yyyy
H:mm:ssz";

  private String     _rssurl;
  private CPContext _cpctx;

  public RSSCPAdaptor()
  {}

  /**
   * Returns the URL of the RSS feed.
   */
  public String getRSSURL()
  {
    return _rssurl;
  }

  /**
   * Sets the URL of the RSS feed.
   */
  public void setRSSURL(String rssurl)
  {
    _rssurl = rssurl;
  }
```

### 4.1.2 Initialized Content Connector

During the initialization of a content connector, Syndication Services builds an instance of the class implementing the `CPAdaptor` interface, in this example of the `RSSCPAdaptor` class, and sets its properties according to the values specified by the administrator during content provider registration.

After the instantiation and restoration of the bean state, the `CPAdaptor.init` method is called to supply an instance of the `CPContext` object to the content connector. The `CPContext` object provides the content connector instance with a set of information that will be useful throughout the duration of the content connector, (see Example 4–2). Such information includes the `CPMessageFactory` object, and which `CPAdaptor` it must use to build data structures that the content connector has to process and return. The content connector is expected to store a reference to the supplied `CPContext` object into one of its fields.

The ping method may be used by the Syndication Services administration code to verify if the content connector has been correctly configured and if it is ready for operation. This RSS content connector could, for example, verify that the RSS URL is active, and correctly return an RSS feed.

***Example 4–2 Initializing the Content Connector***

```
/**
   * Receives and stores the CPContext object from which
   * you can access the CPMessageFactory object.
   */
  public void init(CPContext cpctx)
    throws CPException
  {
    _cpctx = cpctx;
  }
```

```
public boolean ping()
  throws CPException
{
  return true;
}
```

### 4.1.3 Exposed List of Resources

Once a content connector has been installed in the system and one or more content providers have been registered using this content connector, Syndication Services administrators may decide to expose one or more of the content provider resources as offers. To do this, administrators will go through the creation offer wizard and select the content provider resource they want to expose as an offer.

Content connectors expose the list of resources they would like to make available for offer creation through the `CPAdaptor.buildCatalog` API. The content connector will use the `CPMessageFactory` object to build a list of `CPOffer` objects, set the offer metadata, and return an iterator over the list of available resources. Example 4–3 shows how a `CPOffer` is built from an RSS feed. Some of the feed metadata is used to provide defaults for the resource properties. These properties can be reviewed and edited by the administrator at offer creation time.

***Example 4–3 Building a List of CPOffer Objects from an RSS Feed***

```
public Iterator buildCatalog()
  throws CPException
{
  Document docrss = parseRSS();
  Element   elrss = docrss.getDocumentElement();
  NodeList  nlchs = elrss.getElementsByTagName("channel");

 // Loop over the channels defined in this RSS
 // and for each one create an offer.
  CPMessageFactory cpmf = _cpctx.getCPMessageFactory();
  ArrayList      alOffs = new ArrayList();
  for (int i=0; i<nlchs.getLength(); i++) {

    // For each channel, you will create
    // a CPOffer object. CPOffer objects are shown to
    // syndication administrators as content
    // provider resources from which you can build offers.
    Element  elCh = (Element) nlchs.item(i);
    CPOffer cpoff = cpmf.createCPOffer();
```

```
      // Each offer must have a unique ID for its content provider.
      // Set the mandatory ID attribute and any other
      // optional attributes.
      String title = getChildElementValue(elCh, "title");
      if ((title == null) || (title.trim().length() == 0)) {
        title = "unknown";
      }

      // Set the offer properties.
      cpoff.setCPOfferID(title);
      cpoff.setName(title);
      cpoff.setDescription(getChildElementValue(elCh,  "description"));
      cpoff.setRightsHolder(getChildElementValue(elCh, "copyright"));
      cpoff.setProductName("OracleAS Syndication Services RSS Feed Import");
      cpoff.setAtomicUse(false);
      cpoff.setEditable(true);
      cpoff.setShowCredit(false);
      cpoff.setUsageRequired(false);

      alOffs.add(cpoff);
    }

    return alOffs.iterator();
  }
```

Each returned offer must have an offer ID set through the `setCPOfferID` method.

### 4.1.4 Content Packages Built

If a user (subscriber) subscribes to an offer, content updates will be delivered. Content updates are built by calling the `buildPackage` method of the connector associated with the offer for which the content is requested. The `buildPackage` method takes in the following two parameters:

- `CPPackageRequest`: This method provides accessors for parameters related to this content package request. These parameter accessors include the ID of the content provider resource (`CPOfferID`) for which the content package is requested, (this is the same ID set by the content connector in the catalog building using the `CPOffer.setCPOfferID` method), the ID of the subscriber requesting the content package update, and the ID of the subscription associated with this content package request. This method also includes the parameters supplied by the user for this content package request, if any. The content connector will use this set of parameters to determine the resource updates.

- CPRequestContext: This method can be used by the content connector to store resources that it had to reserve or allocate, or both, to build the returned CPPackage object. Such resources can be released in the closePackage package, when the same CPRequestContext instance will be resupplied to the content connector after the content package has been transmitted.

Example 4–4 shows how the sample RSSCPAdaptor class builds a content package. The content connector reads the RSS feed following the URL with which it has been configured. The content connector will then create a CPItem object for each RSS news item found in the feed. The CPItem content is accessible through ContentAccessors methods.

The CPPackage instances and CPItem instance must be created using the CPMessageFactory object supplied to the content connector through the CPContext object. The ContentAccessors methods are simple interface wrappers over Java InputStreams returning the item content. The CPMessageFactory object includes a few factory methods for the most common ContentAccessors interface (including the StringContentAccessor method used in the example). Developers can write their own implementations of the ContentAccessors interface, if needed.

***Example 4–4 Building a Content Package***

```
public CPPackage buildPackage(CPPackageRequest req,
                              CPRequestContext ctx)
  throws CPException
{
  // Parse the RSS document.
  Document docrss = parseRSS();
  Element   elrss = docrss.getDocumentElement();
  Element    elCh = getChannelElement(elrss, req.getOfferID());

  // Create a new content provider package and initialize
  // its properties by using RSS feed values.
  CPMessageFactory cpmf = _cpctx.getCPMessageFactory();
  CPPackage  pkg = cpmf.createCPPackage();

  // Check if the RSS feed has been modified since the
  // the last content package request.
  String oldState = req.getState();
  String newState = getChildElementValue(elCh, "lastBuildDate");
```

```
Date dNewState = getDate(newState);
Date dOldState = null;
if (!oldState.equals(INITIAL_STATE)) {
  dOldState = getDate(oldState);
  if ((dOldState != null) &&
      (dNewState != null) &&
      dOldState.equals(dNewState)) {

    // The RSS feed has not been updated
    // since the last visit. Throw an exception
    // notifying that the content package sequence is
    // up-to-date and no update needs to be reported.
    throw new CPException(CP_PACKAGE_UP_TO_DATE);
  }
  // This is an incremental update over
  // the previous update that was sent.
  // Set the full update flag accordingly.
  pkg.setFullUpdate(false);
}
else {

  // This is a request for a full new update.
  // Set the full update flag accordingly.
  pkg.setFullUpdate(true);
}
pkg.setOldState(oldState);
pkg.setNewState(newState);

// Scan through the RSS news items building
// corresponding CPItem objects.
NodeList nlItems = elCh.getElementsByTagName("item");
for (int i=0; i<nlItems.getLength(); i++) {

  Element elItem = (Element) nlItems.item(i);
  CPItem cpi = cpmf.createCPItem();

  // Sets item file name and content type.
  // Each RSS news item will have a dummy file name
  // and an XML code excerpt for the associated news.
  String itemFilename = "item"+i+".xml";
  cpi.setContentFilename(itemFilename);
  cpi.setContentType("text/xml");

  // Sets the Item ID.
  String guid = getChildElementValue(elItem, "guid");
```

```
    if (guid != null) {
      cpi.setID(guid);
    }

    // Sets the Item name.
    String title = getChildElementValue(elItem, "title");
    if (title != null) {
      cpi.setName(title);
    }
    else {
      // The Item name is mandatory
      // so you need to set it
      // to a value.
      cpi.setName(itemFilename);
    }

    // Sets the Item Activation Date, if any.
    String pubDate = getChildElementValue(elItem, "pubDate");
    if (pubDate != null) {
      Date dPubDate = getDate(pubDate);
      cpi.setActivation( new ISODateTime(dPubDate));
    }

    // Sets the Item content to the RSS Item XML element.
    try {
      StringWriter sw = new StringWriter();
      ((XMLElement) elItem).print( new PrintWriter(sw));
      ContentAccessor ca = cpmf.createContentAccessor(sw.toString());
      cpi.setContent(ca);
    }
    catch(Exception e) {
      throw new CPException(CP_GENERIC_ERROR, e, e);
    }

    // Add the Item to the content package.
    pkg.addPackageItem(cpi);
  }

  return pkg;
}

public void closePackage(CPRequestContext ctx)
  throws CPException
{
}
```

The following properties of the `CPPackage` and its `CPPackageItem` objects are mandatory and must be set by developers in order for the returned content package to be delivered:

- `CPPackage.oldstate` and `CPPackage.newstate`: These properties specify the state of the subscription associated with this resource before and after delivery of this content package.
- `CPItem.ID` and `CPItem.name`: The ID property uniquely identifies the item in the scope of the content package; the named property is a logical name. Developers should also consider setting the `CPItem.subscriptionElement` property, which uniquely identifies the item within the duration of the content provider resource and its associated subscriptions.
- `CPItem.contentFilename:` The content file name property specifies the relative destination of the item, that is, relative to the subscription root, in which the subscription root is defined as a logical directory within which the subscriber will place all file-based content received from the syndicator for that subscription. Paths stated in the content file name must be represented using forward slashes as a path element delimiter.
- `CPItemRef.url`: The URL property is to be used by the end user to retrieve the content.
- `CPItemGroup.ID`: The ID property uniquely identifies the group within the scope of the containing content package.
- `CPItemRemove.subscriptionElement`: This property uniquely identifies the item within the duration of the content provider resource and its associated subscriptions.

**Note:** Propagation of deleted items -- Syndication Services supports the propagation of a deleted item if the content connector that is used supports item removal notification. This can be accomplished by creating `CPItemRemove` objects during the Content Package build operation (see Section 4.1.4 and the API documentation).

When a content provider is registered using a content connector that cannot detect deleted items, removal of any item at the content source will never be revealed to the subscriber and as an effect, such items will be retained on the subscriber site.

For example, using Oracle Application Server Portal as a content source for a content provider (configured on the portal page using a WebDAV content connector), the update of an image item will be seen as the deletion of the old item and the creation of a new one. Because the item deleted will not be propagated to the subscriber, both the old as well the new images will be retained in the subscription.

As a workaround, the content connector can be improved (through custom development using the `CPItemRemove` class) to support item removal notification in the package creation.

### 4.1.5 Incremental Updates

When receiving requests for a new content package update, content providers have the option of returning only the set of items that have been added or modified since the last delivery. To perform incremental updates, the content connector APIs allow for setting a state token that identifies the current state of the resource. When the user requests a content update, the resource's current state will be supplied to the content provider, which will use that state to compute the set of updates to be delivered. The content provider can then deliver the new state identifier in the returned content package.

For example, in a simple implementation, a content provider can decide to store the date of the update as a state identifier. When a content package request is received, the content provider can retrieve the subscription current state of the user by using the `CPPackageRequest.getState()` method, cast it to a date, and then scan the resource to find all the items that have been modified since that date. Then, after completing the scan, building the content package, and returning that package, the content provider will set the state (`CPPackage.setNewState`) to the current date and time.

The sample `RSSCPAdaptor` class uses a similar approach but refers to the RSS feed last modification date as a state identifier. Example 4–5 shows the following state management logic under the `buildPackage` code, and also shows how to handle the special `INITIAL_STATE` state identifier and the property settings of the `fullUpdate` flag for the content package.

***Example 4–5 Performing Incremental Updates***

```
// Check if the RSS feed has been modified since the
// the last content package request.
String oldState = req.getState();
String newState = getChildElementValue(elCh, "lastBuildDate");

Date dNewState = getDate(newState);
Date dOldState = null;
if (!oldState.equals(INITIAL_STATE)) {

  dOldState = getDate(oldState);
  if ((dOldState != null) &&
      (dNewState != null) &&
      dOldState.equals(dNewState)) {

    // The RSS feed has not been updated
    // since the last visit. Throw an exception
    // notifying that the package sequence is
    // up-to-date and no update needs to be reported.
    throw new CPException(CP_PACKAGE_UP_TO_DATE);
  }
  // This is an incremental update over
  // the previous update that was sent.
  // Set the full update flag accordingly.
  pkg.setFullUpdate(false);
}
else {

  // This is a request for a full new update.
  // Set the full update flag accordingly.
  pkg.setFullUpdate(true);
}
pkg.setOldState(oldState);
pkg.setNewState(newState);
```

### 4.1.6 Content Provider Closed

Content connector instances are cached by Syndication Services. When the property value of a content provider is changed by the administrator, the cached instance copy of the content provider is released by the infrastructure database and a new content provider instance is created and configured with the new property values. When releasing a content provider instance, the `CPAdaptor.close` method is called. Developers can use this method to release any resource acquired when using the `init` method.

### 4.1.7 Content Provider Event Push Support

When new content is available from a content provider, an event can be triggered to immediately push this new content to all subscriptions associated with the resource of that content provider. Section 4.1.7.1 is an overview of content provider event push support, and Section 4.1.7.2 describes the event syntax support.

#### 4.1.7.1 Overview

In Syndication Services, content providers can raise an event when a new version of their content is available. This feature complements the time-based content updates described in Section 2.5.4. The conditions that trigger the raising of such an event are specific to each content provider, but they generally imply that a new version of the content associated with a content provider resource is available for delivery. Once such an event is raised, Syndication Services will push a content update to all eligible subscriptions.

The following list shows a possible scenario for a content provider event-based push:

1. The Syndication Services administrator creates an offer offA for a resource rsrcB of content provider cpA.
2. The administrator creates a contract conA, which enables the push delivery rule. A contract conA defines a time period duration of 24 hours for the push delivery rule, and specifies 0 (zero) as the number of updates. In this way, any time a push event is raised by the content provider, there will not be a time period violation, and therefore any event from the content provider will result in a content push to the subscriber.
3. The administrator also grants the offer offA to subscriber sbbU using contract conA.

4. The subscriber sbbU subscribes to an offer offA and gets a new subscription sbtE.
5. The content provider cpA raises a content provider event for its resource rsrcB.
6. Syndication Services pushes a content update to all subscriptions associated with the resource rsrcB of content provider cpA, including the new subscription sbtE.

#### 4.1.7.2 Event Syntax

For Oracle Application Server release 10*g* (9.0.4) and later for the Portal and Wireless mid-tier installation type, a Syndication Services content provider event servlet is installed by default. The URL end point is

```
http://<host>:<port>/syndserver/cpevent
```

The interface Syndication Services provides is based on an HTTP GET request. The syntax is described as follows:

```
http://
<host>:<port>/syndserver/cpevent?cp-id=XXX&cp-offer-id=YYY
```

where `XXX` is the `cp-id` value and `YYY` is the `cp-offer-id` value.

Content providers, by providing the content provider ID (`cp-id`) and content provider offer ID (`cp-offer-id`), can trigger push delivery to all subscriptions that are based on a content provider offer with the given content provider offer ID `cp-offer-id` from the content provider with the given content provider ID `cp-id`.

For example, if the `cp_id` is 801 and the `cp_offer_id` is fcpatest, the URL would appear as:

```
http://<host>:<port>/syndserver/cpevent?cp-id=801&cp-offer-id=fcpatest
```

As an example, using the scenario described in Section 4.1.7.1, the following HTTP GET request will trigger a content provider event push to all subscriptions, including subscription sbtE, associated with rsrcB of content provider cpA:

```
http://syndication_services_host: syndication_services_
port/syndserver/cpevent?cp-id=cpA&cp-offer-id=rsrcB
```

Alternatively, an event can be sent to Syndication Services to push content updates for a list of subscription IDs. The syntax is described as follows:

```
http:// <host>:<port>/syndserver/cpevent?sbt-ids=XX1,XX2,...
```

where XX1 is the first subscription ID, XX2 is the second subscription ID, and so forth. For example:

```
http://<host>:<port>/syndserver/cpevent?sbt-ids=C5E5702058205CC0E0340003BA1906A5
,C5609B440D656B40E0340003BA1906A5
```

## 4.2 Managing Content Connectors

Section 4.2.1 describes how to package and install a new content connector, Section 4.2.2 describes how to list currently installed content connectors, and Section 4.2.3 describes how to uninstall a content connector.

### 4.2.1 Installing a New Content Connector

Content connectors are developed using the Java programming language. To develop a content connector, you have to build a JavaBean, which implements the `CPAdaptor` interface in the `oracle.syndicate.server.cp` package. After the coding has been completed, the content connector code can be compiled and prepared for installation. To compile a content connector class, make sure you include the `syndserver.jar` library in the classpath. The `syndserver.jar` library includes the definition of all the interfaces associated with the development of a content connector. For example, to compile the `RSSCPAdaptor.java` program, you can issue the command shown in Example 4–6.

***Example 4–6 Compiling the RSSCPAdaptor.java File***

```
On UNIX
javac -classpath ${ORACLE_HOME}/lib/xmlparserv2.jar:${ORACLE_
HOME}/syndication/lib/syndserver.jar RSSCPAdaptor.java

On Windows
javac -classpath <ORACLE_HOME>\lib\xmlparserv2.jar:<ORACLE_
HOME>\syndication\lib\syndserver.jar RSSCPAdaptor.java
```

It is recommended that content connector developers perform some unit test of the content connector before installing it in the Syndication Services system. A way to perform a unit test of the content connector is to add a main method in the content connector class, which calls the `buildCatalog` and `buildPackage` APIs. The sample `RSSCPAdaptor.java` program shows how this can be achieved.

Once the content connector code is compiled, build a JAR file containing the compiled classes. Example 4–7 shows the command that builds a JAR file for the RSS content connector compiled previously in Example 4–6.

***Example 4–7 Building an rsscp.jar File***

```
jar cvf rsscp.jar *.class
```

The created `rsscp.jar` file should be copied into a directory where the Syndication Services runtime and administration components can find and load the library when necessary. The default location for content connector libraries is *`${ORACLE_HOME}/syndication/lib/cp`* on UNIX systems or *`<ORACLE_HOME>\syndication\lib\cp`* on Windows systems. Any other library that the connector may depend on, if not already in the Syndication Services classpath, should be copied to this directory. The next step is to copy the JAR file; in a UNIX or Windows environment, the command would be as shown in Example 4–8.

***Example 4–8 Copying the JAR File to the Syndication Services Load Directory***

```
On UNIX
cp rsscp.jar ${ORACLE_HOME}/syndication/lib/cp

On Windows
cp rsscp.jar <ORACLE_HOME>\syndication\lib\cp
```

The new content connector can now be installed in Syndication Services. To complete this step, the command shown in Example 4–9 is executed.

***Example 4–9 Installing the New Content Connector in Syndication Services***

```
On UNIX
java -DORACLE_HOME=${ORACLE_HOME} -jar ${ORACLE_
HOME}/syndication/lib/syndserver.jar -installConnector -name "RSSConnector"
-description "Connector exposing RSS feeds as syndication offers" -className
RSSCPAdaptor

On Windows
java -DORACLE_HOME=<ORACLE_HOME> -jar <ORACLE_
HOME>\syndication\lib\syndserver.jar -installConnector -name "RSSConnector"
-description "Connector exposing RSS feeds as syndication offers" -className
RSSCPAdaptor
```

---

**Note:** On a Solaris operating system, make sure that the LD_LIBRARY_PATH environment variable is set to `<ORACLE_HOME>/lib` before you execute the Java commands shown in Example 4–9.

---

A restart of the Syndication Services runtime component (that is, a restart of the OC4J_Portal instance where the Syndication Services application is deployed) and a restart of the administration component (that is, a restart of Oracle Enterprise Manager) is necessary to let these components detect the new library. Once the content connector has been properly installed and the OC4J containers for the Syndication Services runtime and administration components have been restarted, content providers based on the new content connector can be registered. See Section 2.4 for more information about the content provider registration process.

### 4.2.2 Listing Installed Content Connectors

The `syndserver.jar` library provides a few command line options to show the list of content connectors currently installed in the system and, eventually, uninstall them. Example 4–10 shows the command that returns the list of content connectors that are currently installed for Syndication Services.

***Example 4–10 Listing the Installed Content Connectors***

```
On UNIX
java -DORACLE_HOME=${ORACLE_HOME} -jar ${ORACLE_
HOME}/syndication/lib/syndserver.jar -listConnectors

On Windows
java -DORACLE_HOME=<ORACLE_HOME> -jar <ORACLE_
HOME>\syndication\lib\syndserver.jar -listConnectors

 Installed CPAs
----------------
ID   : 1
Name : FileConnector
Class: oracle.syndicate.server.cp.impl.file.FileCPAdaptor
Description: Content Provider Adaptor fetching content from a file system.

ID   : 2
Name : WebDAVConnector
Class: oracle.syndicate.server.cp.impl.dav.DAVCPAdaptor
Description: Content Provider Adaptor fetching content from repositories
```

```
 accessible through the WebDAV protocol.

ID   : 3
Name : RSS Connector
Class: RSSCPAdaptor
Description: Content Connector for RSS feeds
```

### 4.2.3 Uninstalling a Content Connector

To uninstall a content connector, first check to make sure that no content providers that use this content connector are currently registered. If there are some registered content providers that use this content connector, each must be deleted. Once this condition is met, the content connector can be uninstalled using the command shown in Example 4–11 by supplying the connector ID.

***Example 4–11 Uninstalling the Content Connector Whose Connector ID Is 3***

```
On UNIX
java -DORACLE_HOME=${ORACLE_HOME} -jar ${ORACLE_
HOME}/syndication/lib/syndserver.jar -uninstallConnector 3

On Windows
java -DORACLE_HOME=<ORACLE_HOME> -jar <ORACLE_
HOME>\syndication\lib\syndserver.jar -uninstallConnector 3
```

# A
# Error Messages

This appendix lists and describes the error messages used by Syndication Services to assist administrators and developers. This appendix is divided into four sections: server error messages, client error messages, provider error messages, and ICE error messages.

Runtime errors are recorded in the `application.log` file.

On UNIX, this log file is located under the *`$ORACLE_HOME/j2ee/OC4J_Portal/application-deployments/syndserver/$ISLAND_NUMBER`* directory, while *`$ORACLE_HOME`* is where Oracle Application Server Portal and Oracle Application Server Wireless are installed.

On Windows, this log file is located under the *`<ORACLE_HOME>\j2ee\OC4J_Portal\application-deployments\syndserver\<ISLAND_NUMBER>`* directory, while *`<ORACLE_HOME>`* is where Oracle Application Server Portal and Oracle Application Server Wireless are installed on Windows.

The value for *`$ISLAND_NUMBER`* or *`<ISLAND_NUMBER>`*, for instance, should be something like, "`OC4J_Portal_default_island_1`".

Any errors encountered while using the Oracle Enterprise Manager Oracle Application Server Syndication Services Administrator are returned to the administrator as a message within the GUI.

## A.1 Server Error Messages and Their Descriptions

The following sections describe server error messages.

### A.1.1 General Error Messages

**Error Code: OSS-00120**

*JAXP APIs and XML Parser are not correctly configured.*

**Cause:** A configuration error was reported by the XML Parser during its execution.

**Action:** Verify that `xmlparserv2.jar` library is in the Oracle Application Server classpath and that the classes are available to OC4J.

### A.1.2 Invalid Data Source Init Mode Error Messages

**Error Code: OSS-01002**

*Protocol {0} unsupported.*

**Cause:** The incoming request contains an unsupported protocol.

**Action:** Check the client-side message library and make sure it complies with what Syndication Services supports.

### A.1.3 Database Error Messages

**Error Code: OSS-01100**

*An internal exception occurred. {0}.*

**Cause:** An internal exception occurred, as presented in the exception text.

**Action:** Check the `application.log` file.

On UNIX, this log file is located under the *`$ORACLE_HOME/j2ee/OC4J_Portal/application-deployments/syndserver/$ISLAND_NUMBER`* directory, while *`$ORACLE_HOME`* is where Oracle Application Server Portal and Oracle Application Server Wireless are installed.

On Windows, this log file is located under the *`<ORACLE_HOME>\j2ee\OC4J_Portal\application-deployments\syndserver\<ISLAND_NUMBER>`* directory, while *`<ORACLE_HOME>`* is where Oracle Application Server Portal and Oracle Application Server Wireless are installed on Windows.

The value for *`$ISLAND_NUMBER`* or *`<ISLAND_NUMBER>`*, for instance, should be something like, "`OC4J_Portal_default_island_1`".

If the problem persists, contact Oracle Support Services.

**Error Code: OSS-01102**

*Invalid ID format {0}.*

**Cause:** Internal error. Supplied ID is invalid.

**Action:** ID should be a valid integer.

### A.1.4 Negotiation Error Messages

**Error Code: OSS-01203**

*Syndicator proposes a counter offer.*

**Cause:** Syndicator proposes a counter offer to a subscriber because the supplied offer for the subscription does not conform to what is expected.

**Action:** Review the counter offer and reply in order to proceed with the subscription.

### A.1.5 Database Connection Manager Error Messages

**Error Code: OSS-02100**

*No DataSource initialized.*

**Cause:** No DataSource initialized.

**Action:** Check the infrastructure database and restart the current instance if needed.

**Error Code: OSS-02101**

*Error opening database connection.*

**Cause:** Error opening database connection.

**Action:** Check the infrastructure database status and restart the listener or the database instance, or both, if needed.

**Error Code: OSS-02103**

*Error performing JNDI lookup of {0}.*

**Cause:** Error performing JNDI lookup of the listed entry.

**Action:** Check the `data-sources.xml` file in your OC4J instance `config` directory and make sure the `jdbc/OracleOSS` entry on UNIX or the `jdbc\OracleOSS` entry on Windows exists and contains the correct database connection information.

**Error Code: OSS-02104**

*Error opening OID connection.*

**Cause:** Error opening OID connection.

**Action:** Check the Oracle Internet Directory in the infrastructure instance and restart it if needed.

**Error Code: OSS-02105**

*Missing user/group search base.*

**Cause:** Missing user or group search base.

**Action:** Make sure the Oracle Internet Directory is installed and configured correctly during the infrastructure setup stage.

**Error Code: OSS-02106**

*Syndication mid-tier (version {0}) is incompatible with infrastructure(version {1}), please update the infrastructure.*

**Cause:** Syndication mid-tier is incompatible with infrastructure.

**Action:** Update the infrastructure installation to the latest release.

### A.1.6 Content Connector Error Messages

**Error Code: OSS-03100**

*Error during inspection of CPAdaptor class.*

**Cause:** Failed to inspect CPAdaptor implementation.

**Action:** Check the deployed content connector (CPAdaptor) implementation, and make sure its implementation classes or library is available in the classpath.

**Error Code: OSS-03101**

*The CPAdaptor class was not found.*

**Cause:** The corresponding CPAdaptor class was not found.

**Action:** Make sure the deployed content connector (CPAdaptor) library is located under the `$ORACLE_HOME/syndication/lib/cp` directory on UNIX or under `ORACLE_HOME\syndication\lib\cp` directory on Windows, and restart the Oracle Application Server instance correspondingly.

**Error Code: OSS-03102**

*The CPAdaptor class cannot be instantiated.*

**Cause:** The CPAdaptor class cannot be instantiated.

**Action:** Check the content connector (CPAdaptor) implementation and refer to the Syndication Services documentation, if needed.

**Error Code: OSS-03103**

*The CPAdaptor class cannot be accessed.*

**Cause:** The CPAdaptor class cannot be accessed.

**Action:** Check the content connector (CPAdaptor) implementation and refer to the Syndication Services documentation, if needed.

**Error Code: OSS-03104**

*CPA cannot invoke the target.*

**Cause:** The current content provider adaptor instance cannot invoke the target.

**Action:** Check the content connector (CPAdaptor) implementation and refer to the Syndication Services documentation, if needed.

**Error Code: OSS-03105**

*Security exception.*

**Cause:** Security exception.

**Action:** Check the content connector (CPAdaptor) implementation and refer to the Syndication Services documentation, if needed.

**Error Code: OSS-03106**

*Illegal argument.*

**Cause:** Invalid argument.

**Action:** Check the content connector (CPAdaptor) implementation and refer to the Syndication Services documentation, if needed.

**Error Code: OSS-03107**

*Bad property format.*

**Cause:** Bad property format.

**Action:** Check the content connector (CPAdaptor) configuration and make sure that the values of the connector properties correspond to the right property type.

**Error Code: OSS-03110**

*The value for the property {0} is missing.*

**Cause:** The value for the content connector (CPAdaptor) property listed in the message is missing.

**Action:** Review the content connector (CPAdaptor) configuration and add a value for the offending property.

## A.1.7 Offer Error Messages

**Error Code: OSS-04100**

*Offer {0} is not active.*

**Cause:** The specified offer is not active.

**Action:** Contact the administrator of Syndication Services for more details about the specific offer.

**Error Code: OSS-04101**

*Offer {0} is not granted to the subscriber {1}.*

**Cause:** The specified offer is not granted to the user.

**Action:** The user is trying to a subscribe to an offer not granted to him. Review the offer access settings and modify, if necessary.

**Error Code: OSS-04102**

*{0} is not compliant. Field {1} expecting value {2} while it received {3}.*

**Cause:** The user is trying to subscribe to an offer, however the offer is requesting a subscription that has some differences from the one the user is granted.

**Action:** The user will have to review and change the offer before resubmitting it for subscription. The only offer fields the user can customize in a subscription request are push_url or remove_ice_element.

## A.1.8 Subscription Error Messages

**Error Code: OSS-05101**

*Unrecognized subscriber {0}.*

**Cause:** Unrecognized subscriber. A user with an invalid ID has tried to log in.

**Action:** Check the identity of the subscriber to verify the correct subscriber ID has been sent to the server.

**Error Code: OSS-05103**

*Unrecognized subscription {0}.*

**Cause:** Unrecognized subscription with the listed ID. The user tried to perform an operation on a subscription that does not exist, or whose state is not active.

**Action:** Verify the subscription ID is correct, and if applicable, modify the subscription state to active to let the user perform subscription operations.

**Error Code: OSS-05104**

*Subscription {0} is not active.*

**Cause:** Subscription with the listed ID is not active. The user tried to perform an operation on a subscription that does not exist, or whose state is not active.

**Action:** Verify that the subscription ID is correct, and if applicable, modify the subscription state to active to let the user perform subscription operations.

**Error Code: OSS-05105**

*Subscriber {1} does not own subscription {0}.*

**Cause:** Subscriber with the ID shown does not own the specified subscription.

**Action:** Verify both the subscriber ID and the subscription ID.

**Error Code: OSS-05106**

*Subscription {0} has no delivery rule.*

**Cause:** Subscription with the ID shown has no delivery rule.

**Action:** Verify the subscription ID. Also review the offer being subscribed and make sure its contract contains a valid delivery rule.

**Error Code: OSS-05107**

*No delivery rule allows access to subscription {0} at time {1}.*

**Cause:** A content delivery is requested, but no delivery rules associated with the subscription allow access at this time.

**Action:** Verify the delivery rules associated with the subscriptions. The user will have to retry the delivery request in an appropriate time period.

**Error Code: OSS-05108**

*The URL {0} is not a valid URL.*

**Cause:** The syndicator is trying to push the user a new content package but the user's push URL is not a supported URL.

**Action:** Review the user's push URL for the subscription push delivery rules and, if necessary, notify the user to change the push URL.

**Error Code: OSS-05109**

*Subscription {0} does not have delivery rule {0}.*

**Cause:** Subscription does not have a delivery rule matching the user request.

**Action:** Verify the delivery rules associated with the subscriptions. The user will have to retry the delivery request according to the delivery rules the user has been granted.

**Error Code: OSS-05110**

*Subscription {0} has expired.*

**Cause:** Subscription state has been automatically changed to expired because the expiration policy criteria have been met.

**Action:** Review the subscription state and make sure it is expired.

**Error Code: OSS-05111**

*Subscription {0} has outstanding confirmations.*

**Cause:** The user is requesting a content delivery for a subscription that has outstanding confirmations.

**Action:** The user will have to confirm the previously received in-bound content packages before requesting a new content delivery.

**Error Code: OSS-05112**

*Cannot find package to be confirmed {0} for subscriber {1}.*

**Cause:** The user is trying to confirm a previously delivered content package is unrecognized.

**Action:** Verify the subscription status and the ID of the last content package delivered to the user. Notify the user of the ID so that he can confirm the delivery.

## A.1.9 Scheduler Error Messages

**Error Code: OSS-07001**

*Invalid Scheduler Rule: start date {0} is later than stop date {1}.*

**Cause:** Internal exception, the start date of a push scheduling rule is later than the stop date.

**Action:** Review the contract associated with the subscribed offer and check the push delivery rule regarding the start date and stop date.

**Error Code: OSS-07002**

*Scheduler not initialized: null database provider.*

**Cause:** The Oracle Application Server Syndication Services scheduler component cannot be initialized because the infrastructure database cannot be contacted.

**Action:** Check the infrastructure database status and restart the listener or the database instance, or both, if needed.

**Error Code: OSS-07003**

*Scheduler with ID {0} is down.*

**Cause:** An operation involving the Oracle Application Server Syndication Services scheduler component is needed, but the scheduler status is down.

**Action:** Review the Oracle Application Server Syndication Services system properties and turn on the scheduler component, if needed. A restart of the

OC4J_Portal instance where the Syndication Services application is deployed is required.

**Error Code: OSS-07004**

*Invalid Scheduler Rule: number of updates zero or less.*

**Cause:** A push delivery rule with a number of updates per time period of less than zero cannot be scheduled.

**Action:** Review the number of updates per time period in the Edit Contract page, and change it to a positive number.

### A.1.10 Server Message Related Error Messages

**Error Code: OSS-08781**

*OSS Payload validation failure: invalid value {2} for attribute {1} in message {0}.*

**Cause:** The user submitted a request in an invalid message payload. The message contains invalid attributes.

**Action:** Verify that the user is using the Oracle Application Server Syndication Services client library to develop the application.

**Error Code: OSS-08782**

*OSS Payload validation failure: unexpected element {1} encountered instead of {0}.*

**Cause:** The user submitted a request in an invalid message payload. The message contains an unexpected element.

**Action:** Verify that the user is using the Oracle Application Server Syndication Services client library to develop the application.

**Error Code: OSS-08783**

*OSS Payload validation failure: expected member {0} is missing.*

**Cause:** The user submitted a request in an invalid message payload. The message does not contain an expected element.

**Action:** Verify that the user is using the Oracle Application Server Syndication Services client library to develop the application.

### A.1.11 SQLException Error Messages

**Error Code: OSS-20001**

*There are dependent object(s) {0} that prevent the operation from being completed.*

**Cause:** The Syndication Services metadata repository contains dependent object(s) in the message that prevent the deletion of one of the repository entities.

**Action:** Delete the dependent objects first, and then remove the current object.

**Error Code: OSS-20002**

*{0} dependent object(s) are missing. Operation cannot be completed.*

**Cause:** Expected dependent object(s) are missing from the Syndication Services metadata repository. The backend storage is in an inconsistent state, as the external references of the current object no longer exist.

**Action:** Verify the entities currently existing in the Syndication Services metadata repository, and take the system to a consistent state by removing the offending ones.

**Error Code: OSS-20003**

*The {0} object was not found.*

**Cause:** An object was not found in the Syndication Services metadata repository.

**Action:** Internal server error. Verify the object identity.

**Error Code: OSS-20004**

*{0} cannot be created because there is a duplicate.*

**Cause:** The listed specified entity cannot be created in the Syndication Services metadata repository because it is a duplicate.

**Action:** Avoid duplication when creating a new entity.

**Error Code: OSS-20005**

*Association of {0} and {1} does not exist.*

**Cause:** Internal server error: association of the two specified entities does not exist.

**Action:** Verify the two entities and see if the problem can be resolved.

**Error Code: OSS-20007**

*Association of {0} and {1} already exists.*

**Cause:** Association between the two specified entities already exists.

**Action:** No action required because the requested association has already been performed.

**Error Code: OSS-20008**

*Object {0} not active.*

**Cause:** The specified object is not active.

**Action:** Review the object state and change it to active, if appropriate.

**Error Code: OSS-20009**

*Object {0} with {1} state has expired.*

**Cause:** The specified object state has been changed to expired.

**Action:** Review the subscription state and make sure it is expired.

**Error Code: OSS-20012**

*Offer {0} is not granted to {1}.*

**Cause:** The offer is not granted to the specified user or group.

**Action:** Review the offer access information and grant it to the specified user, if appropriate.

## A.2 Client Error Messages and Their Descriptions

The following are client error messages.

### A.2.1 General Error Messages

**Error Code: OSS-00120**

*JAXP APIs and XML parser are not correctly configured.*

**Cause:** A configuration error was reported by the XML parser during its execution.

**Action:** Verify that `xmlparserv2.jar` library is in the Oracle Application Server classpath and that the classes are available to OC4J.

**Error Code: OSS-00121**

*Unexpected syndication client internal exception encountered.*

**Cause:** An unexpected exception was encountered by the syndication client.

**Action:** Look for the log information associated with the client implementation to get additional error information.

### A.2.2 XML Client State Handler Error Messages

**Error Code: OSS-00150**

*Cannot use file <filename> to save client state because it is a directory.*

**Cause:** The specified file name cannot be used to save the client state because it is a directory.

**Action:** Specify a valid file name for the client state storage that does not conflict with an existing directory.

**Error Code: OSS-00151**

*Cannot use file <filename> to save client state because it cannot be read.*

**Cause:** The specified file name cannot be used to save the client state because it cannot be read.

**Action:** Check that the specified file name for the client state is readable. Grant read permission to the file, if needed.

**Error Code: OSS-00152**

*Cannot create file <filename> to save client state.*

**Cause:** The specified file cannot be created to save the client state.

**Action:** Check that the file can be created in the specified directory. Grant write permission to the directory, if needed.

**Error Code: OSS-00153**

*An error occurred while parsing client state file <filename>.*

**Cause:** The specified file name is not a valid client state file, or it may be corrupted.

**Action:** Verify that the client state storage file exists, is not corrupted, and complies with the state handler used to read it.

**Error Code: OSS-00154**

*Cannot write client state to file <filename>.*

**Cause:** The application encountered an error during the attempt to write the client state information to the specified file.

**Action:** Verify that the client state storage file has been granted write permission.

**Error Code: OSS-00155**

*Cannot add new subscriber with ID <subscriberid> because it is a duplicate.*

**Cause:** The application encountered an error when attempting to add subscriber information to the client state storage file. The file already contains information for a subscriber having the specified ID. Adding a new subscriber with the same ID is not allowed.

**Action:** Use a different subscriber ID for the new subscriber.

**Error Code: OSS-00156**

*Cannot add a new subscription with ID {0} because it is a duplicate.*

**Cause:** The application encountered an error when attempting to add subscription information to the client state storage file. The file already contains information for a subscription having the specified ID. Adding a new subscription with the same ID is not allowed.

**Action:** Use a different subscription ID for the new subscription.

### A.2.3 FileSAXPackageHandler Error Messages

**Error Code: OSS-00160**

*Cannot initialize directory {0}.*

**Cause:** The application encountered an error when attempting to use the specified directory as the storage root destination for the incoming content packages.

**Action:** Check that the specified directory exists, can be accessed by the client application, and has write permission.

**Error Code: OSS-00161**

*Cannot create file for item {0}.*

**Cause:** The application was not able to create a file to store the specified item delivered through a content package.

**Action:** Check that the destination directory has the appropriate write permission, and that the name of the file to be created does not conflict with an existing directory.

**Error Code: OSS-00162**

*Cannot create flush file for item {0}.*

**Cause:** The application was not able to flush the content delivered with the specified item into the destination file.

**Action:** Check that there is enough space in the destination file system to hold the content of the specified item, and that the destination directory has write permission.

**Error Code: OSS-00163**

*Cannot close file for item {0}.*

**Cause:** There was an error on the attempt to close the file output stream for the specified item.

**Action:** Verify that the destination file for the specified item was not used by any other application.

### A.2.4 SyndicateClientServlet Error Messages

**Error Code: OSS-00170**

*Missing value for property {0}.*

**Cause:** The client servlet failed at initialization time when looking for the specified property in the `web.xml` configuration file.

**Action:** Check that the specified property is defined in the `web.xml` file of the application.

**Error Code: OSS-00171**

*Cannot load class {0}.*

**Cause:** The class specified in the `web.xml` file for the SyndDelegator implementation cannot be loaded at runtime.

**Action:** Check that the specified class is available through the OC4J classloader.

### A.2.5 DAVSAXPackageHandler Error Messages

**Error Code: OSS-00180**

*Invalid DAV URL {0}.*

**Cause:** The client application encountered an error during the attempt to connect to the specified URL and to establish WebDAV contact using the specified URL.

**Action:** Make sure that the specified URL is a valid WebDAV end point and that the DAV server is up and running.

**Error Code: OSS-00181**

*Cannot use resource {0} to deliver content package.*

**Cause:** The application encountered an error when attempting to use the specified DAV resource as the storage root destination for the incoming content packages.

**Action:** Check that the specified DAV collection exists, can be accessed by the client application, and the connected user has write permission.

**Error Code: OSS-00182**

*Cannot create resource for item {0}.*

**Cause:** The application was not able to create a DAV resource to store the specified item delivered through a content package.

**Action:** Check that the connected user is granted write permission on the destination collection, and that the name of the resource to be created does not conflict with an existing DAV collection.

**Error Code: OSS-00183**

*Cannot flush resource for item {0}.*

**Cause:** The application was not able to flush the content delivered with the specified item in the destination DAV resource. The reason may be due to limited storage space, or to a permission problem on the destination collection.

**Action:** Check that there is enough space in the destination DAV system to hold the content of the specified item.

**Error Code: OSS-00184**

*Cannot close resource for item {0}.*

**Cause:** There was an error on the attempt to close the resource output stream and to commit the new content delivered with the specified item in the DAV repository.

**Action:** Verify that the destination resource for the specified item was not used by any other application. Check also that the user connected to the DAV repository has the appropriate permission to write and update resources.

**Error Code: OSS-00185**

*Cannot delete resource for item {0}.*

**Cause:** There was an error on the attempt to delete the resource in the DAV repository.

**Action:** Verify that the destination resource for the specified item was not used by any other application. Check also that the user connected to the DAV repository had the appropriate permission to delete resources.

**Error Code: OSS-00186**

*Cannot connect to the DAV end point.*

**Cause:** There was an error on the attempt to establish the HTTP connection to the DAV repository.

**Action:** Verify that the DAV end point is up, the user name and password used to connect are valid, and the authentication system (SSO/OID) is up.

### A.2.6 Syndicate Connection Implementation Error Messages

**Error Code: OSS-00196**

*The state handler is not available.*

**Cause:** The client application cannot use the configured state handler.

**Action:** Check the state handler setting.

**Error Code: OSS-00197**

*Cannot delete subscriber.*

**Cause:** The subscriber could not be deleted because it has some dependent objects that have to be deleted first, or because an error occurred.

**Action:** Check that the state handler system works properly and that all dependent objects have been removed; then, retry.

## A.3 Provider Error Messages and Their Descriptions

The following are provider error messages.

### A.3.1 General Error Messages

**Error Code: OSS-00120**

*JAXP APIs and XML parser are not correctly configured.*

**Cause:** A configuration error was reported by the XML parser during its execution.

**Action:** Verify that `xmlparserv2.jar` library is in the Oracle Application Server classpath and that the classes are available to OC4J.

### A.3.2 SyndicateClientServlet Error Messages

**Error Code: OSS-00170**

*Missing value for property {0}.*

**Cause:** The client servlet failed at initialization time when looking for the specified property in the `web.xml` configuration file.

**Action:** Check that the specified property is defined in the `web.xml` file of the application.

**Error Code: OSS-00171**

*Cannot load class {0}.*

**Cause:** The class specified in the `web.xml` file for the SyndDelegator implementation cannot be loaded at runtime.

**Action:** Check that the specified class is available through the OC4J classloader.

### A.3.3 DAVSAXPackageHandler Error Messages

**Error Code: OSS-00180**

*Invalid DAV URL {0}.*

**Cause:** The client application encountered an error during the attempt to connect to the specified URL and to establish WebDAV contact using the specified URL.

**Action:** Make sure that the specified URL is a valid WebDAV end point and that the DAV server is up and running.

**Error Code: OSS-00181**

*Cannot use resource {0} to deliver content package.*

**Cause:** The application encountered an error when attempting to use the specified DAV resource as the storage root destination for the incoming content packages.

**Action:** Check that the specified DAV collection exists, can be accessed by the client application, and the connected user has write permission.

**Error Code: OSS-00182**

*Cannot create resource for item {0}.*

**Cause:** The application was not able to create a DAV resource to store the specified item delivered through a content package.

**Action:** Check that the connected user is granted write permission on the destination collection, and that the name of the resource to be created does not conflict with an existing DAV collection.

**Error Code: OSS-00183**

*Cannot flush resource for item {0}.*

**Cause:** The application was not able to flush the content delivered with the specified item in the destination DAV resource. The reason may be due to limited storage space, or to a permission problem on the destination collection.

**Action:** Check that there is enough space in the destination DAV system to hold the content of the specified item.

**Error Code: OSS-00184**

*Cannot close resource for item {0}.*

**Cause:** There was an error on the attempt to close the resource output stream and to commit the new content delivered with the specified item in the DAV repository or the timeout value specified is too small.

**Action:** Verify that the destination resource for the specified item was not used by any other application. Check also that the user connected to the DAV repository has the appropriate permission to write and update resources and has specified a large enough timeout value for the `oracle.syndicate.ui.provider.ProviderSyndDelegator.TimeoutSyndConn` initialization parameter. See *Oracle Application Server Portal Configuration Guide* for more information.

**Error Code: OSS-00185**

*Cannot delete resource for item {0}.*

**Cause:** There was an error on the attempt to delete the resource in the DAV repository.

**Action:** Verify that the destination resource for the specified item was not used by any other application. Check also that the user connected to the DAV repository had the appropriate permission to delete resources.

**Error Code: OSS-00186**

*Cannot connect to the DAV end point.*

**Cause:** There was an error on the attempt to establish the HTTP connection to the DAV repository.

**Action:** Verify that the DAV end point is up, the user name and password used to connect are valid, and the authentication system (SSO/OID) is up.

### A.3.4 ProviderSyndRequestHandler Error Messages

**Error Code: OSS-00206**

*Cannot find channel ID for the push request.*

**Cause:** The push request cannot be completed because the channel ID was not correctly specified in the request URL.

**Action:** Verify that the server is pushing content using the URL originally created at subscription time.

**Error Code: OSS-00207**

*Cannot find the WebDAV URL for the requested channel ID {0}.*

**Cause:** The provider application cannot retrieve the DAV URL setting to connect to the content repository.

**Action:** Check that provider settings are correctly entered. Use the edit default page to check or update those settings.

**Error Code: OSS-00208**

*Cannot find the connection for the current transaction.*

**Cause:** The application failed to get the connection to commit the current transaction associated with the push process.

**Action:** Restart the provider Web application. If the problem persists contact the administrator.

## A.4 ICE Error Messages and Their Descriptions

The following are ICE error messages.

### A.4.1 3*nn*: Payload-Level Status Messages

**Error Code: ICE-300**

*Generic catastrophic payload error.*

**Cause:** A generic status code indicating an inability to comprehend the received payload.

**Action:** Check the request payload and resend it; if the problem persists, have your Syndication Services administrator check the status of the requester's account.

**Error Code: ICE-301**

*Payload incomplete or cannot parse.*

**Cause:** The payload sent is severely garbled and cannot be parsed, possibly because it is not an XML payload. Expecting an XML payload and received a payload that was not XML.

**Action:** Resend the payload and be sure it is an XML payload.

**Error Code: ICE-302**

*Payload not well formed XML.*

**Cause:** The payload sent is recognizable as XML, but is not well formed per the definition of XML.

**Action:** Ask your Syndication Services administrator to fix the corresponding syndication operation module.

**Error Code: ICE-303**

*Payload validation failure.*

**Cause:** The payload failed validation according to the document type definition (DTD).

**Action:** Check the request payload. It must conform to the specified request DTD provided by Syndication Services.

**Error Code: ICE-304**

*Temporary responder problem.*

**Cause:** The responder may be too busy, or an update may be in progress, and so forth. Eventually an identical retry request might succeed.

**Action:** Check the requester's local message system for any notification of system downtime, or try again later.

**Error Code: ICE-320**

*Incompatible version.*

**Cause:** The communication protocol used in the request is not supported.

**Action:** If you use the Syndication Services client library, check the list of protocol versions it supports and then contact Oracle Support Services. If you do not use the Syndication Services client library, contact your local system administrator for help.

**Error Code: ICE-331**

*Failure fetching external data.*

**Cause:** The receiver could not follow an external reference (URL) given to it by the sender as an external entity reference. Syndication Services must not reply with this code.

**Action:** Double-check the content offer reference link and all of its related access policies, such as certificates for an HTTP or HTTPS approach.

**Error Code: ICE-390**

*Payload temporary redirect.*

**Cause:** The payload is being redirected temporarily using the supplied new transport communication end point. In ICE or HTTP, this means a new URL is supplied.

**Action:** Contact Oracle Support Services.

**Error Code: ICE-391**

*Payload permanent redirect.*

**Cause:** The payload is being redirected permanently using the supplied new transport end point. In ICE or HTTP, this means a new URL is supplied.

**Action:** Contact Oracle Support Services.

### A.4.2 4*nn*: Request-Level Status Messages

**Error Code: ICE-400**

*Generic request error.*

**Cause:** A generic status code indicating an inability to comprehend the request.

**Action:** Check the request payload for errors and resend it.

**Error Code: ICE-401**

*Incomplete or cannot parse.*

**Cause:** The request sent is severely garbled and cannot be parsed.

**Action:** Check the request payload for errors, and also make sure you close the communication channel only after the request is completely sent out.

**Error Code: ICE-402**

*Not well formed XML.*

**Cause:** The request sent is recognizable XML, but is not well formed per the definition of XML.

**Action:** Check the request payload for errors.

**Error Code: ICE-403**

*Validation failure.*

**Cause:** The request failed validation according to the DTD.

**Action:** Make sure the request payload conforms to the request DTD provided by Syndication Services.

**Error Code: ICE-405**

*Unrecognized sender.*

**Cause:** The sender does not have permission to access the current server.

**Action:** Contact your Syndication Services administrator to verify sender information.

**Error Code: ICE-406**

*Unrecognized subscription.*

**Cause:** Either the subscription ID is not correct, or the subscription is not in a consistent state.

**Action:** Contact your Syndication Services administrator to verify sender information.

**Error Code: ICE-407**

*Unrecognized operation.*

**Cause:** The request type is not supported at the server side.

**Action:** Make sure the operation you want to perform is supported for Syndication Services.

**Error Code: ICE-408**

*Unrecognized operation arguments.*

**Cause:** The request argument is not supported at the server side.

**Action:** Check the request payload against the request DTD provided by Syndication Services. If the error persists, contact Oracle Support Services.

**Error Code: ICE-409**

*Not available under this subscription.*

**Cause:** The requester has referred to something not covered by the subscription referenced in the request.

**Action:** Resubscribe with additional conditions.

**Error Code: ICE-410**

*Not found.*

**Cause:** A generic error for being unable to find something.

**Action:** Check the error message for the detailed description.

**Error Code: ICE-411**

*Unrecognized package sequence state.*

**Cause:** The package sequence identifier supplied by the sender is not understood by the receiver.

**Action:** Check the payload for errors. If the error persists, contact Oracle Support Services.

**Error Code: ICE-412**

*Unauthorized.*

**Cause:** The subscriber does not have permission to access Syndication Services or a certain set of content providers.

**Action:** Contact your Syndication Services administrator.

**Error Code: ICE-413**

*Forbidden.*

**Cause:** Access violation according to the subscriber's profile.

**Action:** Contact your Syndication Services administrator.

**Error Code: ICE-414**

*Business term violation.*

**Cause:** The current operation or the content of the current request does not comply with the business terms between the subscriber and Syndication Services.

**Action:** Review the business terms for the subscriber account and adjust the request.

**Error Code: ICE-420**

*Constraint failure.*

**Cause:** The subscriber's request for content occurred from Syndication Services violated or was inconsistent with the constraints specified in the subscriptions.

**Action:** Review the constraint for the subscription and adjust the subscriber's request payload.

**Error Code: ICE-422**

*Schedule violation, try again later.*

**Cause:** The request was made at an incorrect time.

**Action:** Issue the request for a package update within the agreed-upon timing window.

**Error Code: ICE-430**

*Not confirmed.*

**Cause:** A generic error indicating the operation is not confirmed.

**Action:** Check the subscriber's local system setting to send confirmations. If the error persists, contact Oracle Support Services.

**Error Code: ICE-431**

*Error fetching external data.*

**Cause:** The receiver could not follow an external reference (URL) given to it by the sender.

**Action:** Check the subscriber's local system settings to enable such action.

**Error Code: ICE-440**

*Sorry.*

**Cause:** As part of the negotiation process, this response indicates the proposal is unacceptable for some reason.

**Action:** Review the subscription request and make necessary modifications; then, resend the request.

**Error Code: ICE-441**

*Counter proposal.*

**Cause:** As part of the negotiation process, this response indicates an updated offer or a counter proposal.

**Action:** Review the counter offer subscription request and make necessary modifications, then, send the response.

**Error Code: ICE-442**

*Renegotiation in progress.*

**Cause:** As part of the renegotiation process, this response indicates that renegotiation is in progress.

**Action:** Proceed to the renegotiation process.

**Error Code: ICE-443**

*Offer acknowledged but deferred.*

**Cause:** Used during parameter negotiation. The receiver of an offer cannot evaluate the offer without human intervention. The sender may try again later with the offered parameters and the same subscription ID. The receiver is expected to respond with a new offer (reflecting the result of the intervention) at some later time.

**Action:** After waiting for a reasonable period of time, if there is no response, contact your Syndication Services administrator to see if there was a network problem or system problem, which might explain why no response was received. If there were no obvious network or system problems, then the receiver may either contact the sender directly and come to some agreement, at which time the sender will send out a new offer, or the sender may resend the previous offer at a later time.

**Error Code: ICE-450**

*Range type invalid.*

**Cause:** Used during parameter negotiation. A protocol-specific range of type numerical, lexical, time, or enumeration, has been incorrectly specified.

**Action:** Review the corresponding sections inside the request payload and make corrections.

**Error Code: ICE-451**

*Span selection out of range.*

**Cause:** Used during parameter negotiation. The value selected in a span is less than the minimum or greater than the maximum.

**Action:** Review the request payload and make the correction.

**Error Code: ICE-452**

*Selection error.*

**Cause:** Used during parameter negotiation. The number of items selected in an enumeration does not match the select attribute. For example, the select attribute may require one or more items to be selected but the number selected in the response was zero.

**Action:** Review the request payload and make the correction.

**Error Code: ICE-453**

*Enumeration selection empty or invalid.*

**Cause:** Used during parameter negotiation. The number of selectable `ice-enum-item` elements is not compatible with the `select` attribute. For example, the `select` attribute might indicate that at least one `ice-enum-item` element must be selected, but the actual number of items available is less than the number of `ice-enum-item` elements in the enumeration.

**Action:** Review the request payload and make the correction.

**Error Code: ICE-454**

*Span invalid.*

**Cause:** Used during parameter negotiation. A span has values that violate its structure. For example, a span point must not have a value less than the span minimum, nor larger than the span maximum.

**Action:** Review the request payload and make the correction.

**Error Code: ICE-456**

*Time formats incompatible for comparison.*

**Cause:** Used during parameter negotiation. Two time values to be compared during negotiation are in incompatible formats. For example, a duration is being compared with an ICE date or time.

**Action:** Review the request payload and make the correction.

### A.4.3 5*nn*: Implementation Error and Operational Failure Messages

**Error Code: ICE-500**

*Generic internal responder error.*

**Cause:** This is a catch-all response for general problems.

**Action:** Contact your Syndication Services administrator.

**Error Code: ICE-501**

*Temporary responder problem.*

**Cause:** The responder may be too busy, or an update may be in progress, and so forth.

**Action:** Eventually, an identical retry request might succeed.

**Error Code: ICE-503**

*Not implemented.*

**Cause:** The service does not implement the requested operation.

**Action:** Make sure the operation you want to perform is supported by Syndication Services.

### A.4.4 6*nn*: Pending State Status Messages

**Error Code: ICE-601**

*Unsolicited message must be processing now.*

**Cause:** The syndicator has unsolicited messages to send to the subscriber, and the subscriber has not yet requested them.

**Action:** Issue the request for pending packages to Syndication Services.

**Error Code: ICE-602**

*Excessive confirmations outstanding.*

**Cause:** The syndicator had requested confirmation of package delivery, and now refuses to perform any additional operations until the subscriber supplies the confirmations (positive or negative).

**Action:** Issue the confirmation for the requested package delivery.

**Error Code: ICE-603**

*No more confirmations to send.*

**Cause:** Syndication Services requests a package delivery confirmation that the subscriber believes has already been sent out.

**Action:** No action is needed.

**Error Code: ICE-604**

*No more unsolicited messages.*

**Cause:** The subscriber sent an `ice-unsolicited-now` message, but the syndicator has no unsolicited messages to send.

**Action:** Check the subscriber's local system; if the inconsistencies still persist, contact Oracle Support Services.

B

# Syndication Services Security

This appendix describes the architecture and configuration of security for Syndication Services. This appendix covers the following topics:

- About Syndication Services Security
- Configuring Syndication Services Security

## B.1 About Syndication Services Security

Syndication Services user support is achieved through the use of the LDAP-based provider and the OC4J Java Authentication and Authorization Service. This section covers the following topics:

- Protecting Syndication Services Resources
- Managing and Enforcing Protected Syndication Services Resources
- Using Oracle Application Server Security Services

See Section 2.3 for more information about user management.

### B.1.1 Protecting Syndication Services Resources

Syndication Services security protects the following resources:

- Data -- protected write access to the data stored in the Syndication Services repository; this is typically metadata of Syndication Services.
- Functions -- administrative operations to the Syndication Services repository.
- Passwords -- passwords and other security-related data including:
  - Content provider property values

- User's authentication information for push URLs
- Password of the Portal syndication user

See Section 2.3 for more information about Syndication Services user management. See *Oracle Application Server Portal Configuration Guide* for more information about the Portal syndication user and syndicating content into Oracle Application Server Portal.

### B.1.2 Managing and Enforcing Protected Syndication Services Resources

JAZN (the Oracle implementation of Java Authentication and Authorization Service (JAAS)) and the Syndication Services application manage and enforce write access to the data stored in the Syndication Services repository. JAZN determines the identity and the security role of a user. Only the owner has rights to update data. See Section 2.3 for more information about Syndication Services user management.

For administrative operations, Syndication Services administration is managed by Oracle Enterprise Manager; Oracle Enterprise Manager protects the servlets that provide administrative operations.

Passwords and other security-sensitive information, persistently stored in the database, are further protected by the database `DBMS_OBFUSCATION` PL/SQL package.

### B.1.3 Using Oracle Application Server Security Services

Syndication Services optimizes the JAZN user-level security features, and uses secure socket layer (SSL) encryption both on the server side and on the client side, for accessing Oracle Application Server infrastructure database options.

## B.2 Configuring Syndication Services Security

Section B.2.1 and Section B.2.2 describe configuring Syndication Services security.

### B.2.1 Syndication Services Repository

To ensure the confidentiality of the communication between Syndication Services and the clients:

1. Configure Oracle HTTP Server (OHS)/SSL listener to provide HTTPS access.
2. Configure OC4J to prohibit HTTP access.

The configuration should be done to all security-sensitive Syndication Services end points.

### B.2.2 Syndication Services Client

If you use the provided Syndication Services client library to develop applications to communicate with the Syndication Services repository, you can set connection properties, such as credentials to be used in case of an HTTPS connection and the HTTP proxy to be used (see Section 3.1). You can also use the Oracle Enterprise Manager security features to configure various HTTP transport security properties. See *Oracle Enterprise Manager Basic Installation and Configuration* and *Oracle Enterprise Manager Advanced Configuration* for more information.

# C Sample Syndication Client

This appendix provides a listing of the `SampleSyndicationClient.java` program. See Section C.1.

## C.1 Listing of the SampleSyndicationClient.java Program

A listing of the `SampleSyndicationClient.java` program follows:

```
import java.util.*;

import oracle.syndicate.*;
import oracle.syndicate.client.*;
import oracle.syndicate.client.handler.file.*;

import oracle.syndicate.message.*;
import oracle.syndicate.message.pkg.*;
import oracle.syndicate.message.offer.*;
import oracle.syndicate.message.response.*;
import oracle.syndicate.message.response.event.*;

/**
 *  @version $Header: SampleSyndicationClient.java 07-mar-2003.19:12:51 mcarrer Exp $
 *  @author  mcarrer
 *  @since   release specific (what release of product did this appear in)
 */
public class SampleSyndicationClient
{
  private SyndicateConnection _sc;

  public SampleSyndicationClient()
  {
  }
```

```
public static void main(String args[])
  throws Exception
{
  if (args.length < 3) {
    System.err.println("java SampleSyndicationClient ossurl username password");
    System.exit(0);
  }

  SampleSyndicationClient ssc = new SampleSyndicationClient();

  //
  // Step 0: Initialize syndication connection
  // and ping to the server to make sure it
  // up and running, and we can authenticate.
  //
  ssc.initConnection(args);
  ssc.ping();

  //
  // Step 1: Get the catalog of offers available
  // and select the first offer appearing in the catalog.
  //
  Catalog cat = ssc.getCatalog();
  Offer   off = ssc.getFirstOffer(cat);

  //
  // Step 2: Subscribe to the selected offer.
  //
  Subscription sbt = ssc.subscribe(off);

  //
  // Step 3: Get the content package associated
  // with the new subscription. If necessary,
  // confirm to the syndication server the
  // receipt of the content package.
  //
  //SyndicatePackage pkg = ssc.getPackage(sbt);
  //ssc.confirmPackage(pkg);

  //
  // Step 4: Verify the subscription status by
  // checking subscription expiration criteria,
  // such as the quantity remaining.
  //
```

```
    ssc.status(sbt);

    //
    // Step 5: Get session events.
    //
    ssc.events(sbt);

    //
    // Step 6: Cancel subscription.
    //
    ssc.cancel(sbt);
  }


  /**
   * Creates a SyndicateConnection using
   * an XML state handler storing client state
   * in the synd-client.xml file in the current directory.
   */
  private void initConnection(String argv[])
    throws SyndicateException
  {
    // Acquire a SyndicateConnectionFactory,
    // optionally set the connection parameters,
    // such as proxy info, timeout, and credentials.
    SyndicateConnectionFactory scf = SyndicateConnectionFactory.getInstance();
    // scf.setTimeout(1000);
    // scf.setProxy("myproxyhost", iMyProxtPort);

    // Create a default XML state handler storing subscription state in a file.
    SyndicateClientStateHandler scsh =
 scf.getDefaultSyndicateClientStateHandler("synd-client.xml");
    _sc = scf.createSyndicateConnection(argv[0],  // url of the oss server
                                        argv[1],  // username
                                        argv[2],  // password
                                        scsh);
  }

  private void ping()
    throws SyndicateException
  {
    Response resp = _sc.ping();
    System.out.println("ping: "+resp.getCode().getPhrase());
  }
```

```
private Catalog getCatalog()
  throws SyndicateException
{
  return _sc.getCatalog();
}

private Offer getFirstOffer(Catalog cat)
  throws SyndicateException
{
  // Iterates through the catalog and returns the
  // first offer encountered.
  Offer   off = null;
  Iterator it = cat.getCatalogItems();
  while (it.hasNext() && (off == null)) {
    CatalogItem item = (CatalogItem)it.next();
    if (item.getCatalogItemType() == CatalogItem.OFFER) {

      off = (Offer) item;
    }
    else {
      // You got an offer group.
      off = getFirstOffer((OfferGroup) item);
    }
  }

  // Print a few offer details.
  System.out.println("offer: "+off.getID()+" - "+off.getDescription());
  DeliveryPolicy dp = off.getDeliveryPolicy();
  if (dp.getStartDate() != null) {
    System.out.println("\t start date: "+dp.getStartDate());
  }
  if (dp.getStopDate() != null) {
    System.out.println("\t  stop date: "+dp.getStopDate());
  }
  Iterator dlrs = dp.getDeliveryRules();
  while (dlrs.hasNext()) {
    DeliveryRule dlr = (DeliveryRule) dlrs.next();
    System.out.println("\t  dlr mode: "+dlr.getMode());
  }
  return off;
}


private Offer getFirstOffer(OfferGroup offgrp)
  throws SyndicateException
```

```
{
  Offer   off = null;
  Iterator it = offgrp.getCatalogItems();
  while (it.hasNext() && (off == null)) {

    CatalogItem item = (CatalogItem)it.next();
    if (item.getCatalogItemType() == CatalogItem.OFFER) {

      off = (Offer) item;
    }
    else {
      // You got an offer group.
      off = getFirstOffer((OfferGroup) item);
    }
  }
  return off;
}

private Subscription subscribe(Offer off)
  throws SyndicateException
{
  // If the offer contains a push delivery rule,
  // you can use the offer APIs to set the destination URL.
  DeliveryPolicy dp = off.getDeliveryPolicy();
  Iterator   itdlrs = dp.getDeliveryRules();
  while (itdlrs.hasNext()) {

    DeliveryRule dlr = (DeliveryRule) itdlrs.next();
    if (DeliveryRule.DELIVERY_RULE_MODE_PUSH.equals(dlr.getMode())) {

      dlr.setURL("http://mysyndicationclient.com/syndclient/listener");
      // Optionaly set username/password.
      // dlr.setPushAuthUsername("me");
      // dlr.setPushAuthUsername("pwd");
      // If raw content is requested (for example, a mailto or
      // ftp url has been supplied), set the raw content flag.
      // dlr.setRawFormatSupport(true);
    }
  }

  Subscription sbt = _sc.subscribe(off);
  System.out.println("subscription: "+sbt.getSubscriptionID());
  return sbt;
}
```

```
  private SyndicatePackage getPackage(Subscription sbt)
    throws SyndicateException
  {
    // Use a FileSAXPackageHandler so that
    // syndicate content will be stored in
    // the local file system /tmp directory.
    FileSAXPackageHandler fsph = FileSAXPackageHandler.getInstance("/tmp/");

    // This will get content incremented since the last update.
    // The current subscriber state will be fetched from
    // the SyndicateClientStateHandler used by this connection.
    // To request a full update of the content versus an incremental
    // one, use the following syntax:
    // _sc.getPackage(sbt.getSubscriptionID(),
    //                SyndicateSubscription.STATE_ICE_INITIAL,
    //                null,
    //                fsph);
    SyndicatePackage sp = _sc.getPackage(sbt.getSubscriptionID(), null, fsph);
    return sp;
  }

  private void confirmPackage(SyndicatePackage pkg)
    throws SyndicateException
  {
    // Check if the package requires confirmation.
    // If so, confirm it.
    if (pkg.hasConfirmation()) {

      Response resp = _sc.confirm(pkg.getID());
      System.out.println("confirmed "+pkg.getID()+":
 "+resp.getCode().getPhrase());
    }
  }

  private void status(Subscription sbt)
    throws SyndicateException
  {
    Status sts = _sc.getStatus(sbt.getSubscriptionID());

    Subscription sbtNew = (Subscription) sts.getSubscriptions().next();
    System.out.println(" sbt "+sbtNew.getSubscriptionID()+":");
    System.out.println("     expiration priority: "+
                     Subscription.EXPIRATION_PRIORITIES[sbtNew.getExpirationPriority()]);
    System.out.println("     expiration date: "+sbtNew.getExpirationDate());
    System.out.println("     quantity remaining: "+sbtNew.getQuantityRemaining());
  }

  private void cancel(Subscription sbt)
    throws SyndicateException
  {
    Cancellation canc = _sc.cancelSubscription(sbt.getSubscriptionID(),
                                              "boring", "en_us");

    System.out.println("cancelled: "+canc.getCancellationID());
  }


  private void events(Subscription sbt)
    throws SyndicateException
  {
    Events evt = _sc.getEvents(null, null, sbt.getSubscriptionID());
    EventLog evtlog = evt.getEventLog();
    Iterator itEvts = evtlog.getEventItems();
    while (itEvts.hasNext()) {

      EventItem ei = (EventItem) itEvts.next();
      if (ei.getType() ==  EventItem.EVENT_TYPE_MSG) {

        EventMsg evtmsg = (EventMsg) ei;
        System.out.println(evtmsg.getRequestStart()+" "
                           +evtmsg.getRequest()+" "+
                           evtmsg.getResponseType());
      }
    }
  }
}
```

D

# RSS Content Connector (CPAdaptor)

This appendix provides a listing of the `RSSCPAdaptor.java` program in Section D.1.

## D.1 Listing of the RSSCPAdaptor.java Program

A listing of the `RSSCPAdaptor.java` program follows:

```
import java.io.*;
import java.net.*;
import java.text.*;
import java.util.*;

import org.w3c.dom.*;
import javax.xml.parsers.*;
import oracle.xml.parser.v2.*;

import oracle.syndicate.message.pkg.*;
import oracle.syndicate.server.cp.*;
import oracle.syndicate.server.cp.message.*;
import oracle.syndicate.server.cp.impl.exmsgs.*;

import oracle.syndicate.util.date.*;

// For testing only.
import oracle.syndicate.util.io.*;
import oracle.syndicate.server.cp.impl.message.*;

/**
 * Content provider adaptor (content connector) creates a
 * CPOffer for each channel defined in a given RSS feed. CPOffers will be
 * shown to Syndication Services administrators as resources, for which they can create offers.
 * RSS (Really Simple Syndication) is a mechanism for enabling
 * lightweight syndication of content.
 * More details on RSS can be found at: http://backend.userland.com/rss.
 * The adaptor exposes only one property:
 * the url of the RSS feed for which the offer has to be created.
 * Custom exception messages can be supplied by subclassing the CPException
 * class.
 * The adaptor code can be compiled as follows:
 *   javac -d demo/cp/rss/classes -classpath lib/syndserver.jar:../../xdk/lib/xmlparserv2.jar
demo/cp/rss/src/RSSCPAdaptor.java
 * Standalone unit tests can be run by:
 *   java -Dhttp.proxyHost=www-proxy.us.oracle.com -Dhttp.proxyPort=80 -classpath
lib/syndserver.jar:../../xdk/lib/xmlparserv2.jar:lib/redist/jazn/jazn.jar:demo/cp/rss/classes/
RSSCPAdaptor http://static.userland.com/gems/backend/rssTwoExample2.xml "Scripting News"
 *
 * To install:
 *   java -DORACLE_HOME=$OH -jar lib/syndserver.jar -installConnector -name "RSS Content
Connector" -description "Allows to create offers based on RSS feeds" -className RSSCPAdaptor
 *
 *  @version $Header: RSSCPAdaptor.java 03-apr-2003.13:23:05 mcarrer Exp $
 *  @author  mcarrer
 *  @since   release specific (what release of product did this appear in)
 */
public class RSSCPAdaptor
  implements CPAdaptor, OSSExceptionConstants
{
  private static final String LAST_BUILD_DATE_FORMAT = "EEE, dd MMM yyyy H:mm:ss z";

  private String    _rssurl;
  private CPContext _cpctx;

  public RSSCPAdaptor()
  {}

  /**
   * Returns the URL of the RSS feed.
   */
  public String getRSSURL()
  {
    return _rssurl;
  }

  /**
   * Sets the URL of the RSS feed.
   */
```

```
public void setRSSURL(String rssurl)
{
  _rssurl = rssurl;
}

/**
 * Receives and stores the CPContext from which
 * you can access the CPMessageFactory object.
 */

public void init(CPContext cpctx)
  throws CPException
{
  _cpctx = cpctx;
}

public boolean ping()
  throws CPException
{
  // TODO
  return true;
}

/**
 * Returns only one offer corresponding to the RSS content feed.
 */
public Iterator buildCatalog()
  throws CPException
{
  Document docrss = parseRSS();
  Element   elrss = docrss.getDocumentElement();
  NodeList  nlchs = elrss.getElementsByTagName("channel");

  // Loop over the channels defined in this RSS
  // and for each one, create an offer.
  CPMessageFactory cpmf = _cpctx.getCPMessageFactory();
  ArrayList      alOffs = new ArrayList();
  for (int i=0; i<nlchs.getLength(); i++) {

    // For each channel, you will create
    // a CPOffer. CPOffers are shown to
    // Syndication administrators as content
    // provider resources from which you can build offers.
    Element  elCh = (Element) nlchs.item(i);
    CPOffer cpoff = cpmf.createCPOffer();
```

```
      // Each offer must have a unique ID for its content provider.
      // Set the mandatory ID attribute and any other
      // optional attributes.
      String title = getChildElementValue(elCh, "title");
      if ((title == null) || (title.trim().length() == 0)) {
        title = "unknown";
      }

      // Set the offer properties.
      cpoff.setCPOfferID(title);
      cpoff.setName(title);
      cpoff.setDescription(getChildElementValue(elCh,  "description"));
      cpoff.setRightsHolder(getChildElementValue(elCh, "copyright"));
      cpoff.setProductName("OracleAS Syndication Services RSS Feed Import");
      cpoff.setAtomicUse(false);
      cpoff.setEditable(true);
      cpoff.setShowCredit(false);
      cpoff.setUsageRequired(false);

      alOffs.add(cpoff);
    }

    return alOffs.iterator();
  }

  public CPPackage buildPackage(CPPackageRequest req,
                                CPRequestContext ctx)
    throws CPException
  {
    // Parse the RSS document.
    Document docrss = parseRSS();
    Element   elrss = docrss.getDocumentElement();
    Element    elCh = getChannelElement(elrss, req.getOfferID());

    // Create a new content package and initialize
    // its properties by using RSS feed values.
    CPMessageFactory cpmf = _cpctx.getCPMessageFactory();
    CPPackage  pkg = cpmf.createCPPackage();

    // Check if the RSS feed has been modified since the
    // the last content package request.
    String oldState = req.getState();
    String newState = getChildElementValue(elCh, "lastBuildDate");
```

```
Date dNewState = getDate(newState);
Date dOldState = null;
if (!oldState.equals(INITIAL_STATE)) {

  dOldState = getDate(oldState);
  if ((dOldState != null) &&
      (dNewState != null) &&
      dOldState.equals(dNewState)) {

    // The RSS feed has not been updated
    // since the last visit. Throw an exception
    // notifying that the content package sequence is
    // up-to-date and no update needs to be reported.
    throw new CPException(CPExceptionConstants.CP_PACKAGE_UP_TO_DATE);
  }
  // This is an incremental update over
  // the previous update that was sent.
  // Set the full update flag accordingly.
  pkg.setFullUpdate(false);
}
else {

  // This is a request for a full new update.
  // Set the full update flag accordingly.
  pkg.setFullUpdate(true);
}
pkg.setOldState(oldState);
pkg.setNewState(newState);

// Scan through the RSS items, building
// corresponding CPItem objects.
NodeList nlItems = elCh.getElementsByTagName("item");
for (int i=0; i<nlItems.getLength(); i++) {

  Element elItem = (Element) nlItems.item(i);
  CPItem cpi = cpmf.createCPItem();

  // Sets item file name and content type.
  // Each RSS news item will have a dummy file name
  // and an XML code excerpt for the associated news.
  String itemFilename = "item"+i+".xml";
  cpi.setContentFilename(itemFilename);
  cpi.setContentType("text/xml");

  // Sets the item ID.
```

```
      String guid = getChildElementValue(elItem, "guid");
      if (guid != null) {
        cpi.setID(guid);
      }

      // Sets the item name.
      String title = getChildElementValue(elItem, "title");
      if (title != null) {
        cpi.setName(title);
      }
      else {
        // Item name is mandatory,
        // so you need to set it
        // to a value.
        cpi.setName(itemFilename);
      }

      // Sets the item activation date, if any.
      String pubDate = getChildElementValue(elItem, "pubDate");
      if (pubDate != null) {
        Date dPubDate = getDate(pubDate);
        cpi.setActivation( new ISODateTime(dPubDate));
      }

      // Set the item content to the RSS item XML element.
      try {
        StringWriter sw = new StringWriter();
        ((XMLElement) elItem).print( new PrintWriter(sw));
        ContentAccessor ca = cpmf.createContentAccessor(sw.toString());
        cpi.setContent(ca);
      }
      catch(Exception e) {
        throw new CPException(CPExceptionConstants.CP_GENERIC_ERROR, e, e);
      }

      // Add the item to the content package.
      pkg.addPackageItem(cpi);
    }

    return pkg;
  }

  public void closePackage(CPRequestContext ctx)
    throws CPException
  {
}

private Document parseRSS()
  throws CPException
{
  Document doc = null;
  InputStream is = null;
  try {

    URL url = new URL(_rssurl);
    URLConnection urlc = url.openConnection();
    is = urlc.getInputStream();

    DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
    javax.xml.parsers.DocumentBuilder db = dbf.newDocumentBuilder();
    doc = db.parse(is);
  }
  catch (Exception e) {
    throw new CPException(CPExceptionConstants.CP_GENERIC_ERROR, e, e);
  }
  finally {
    if (is != null) {
      try { is.close(); }
      catch (Exception e) {
        throw new CPException(CPExceptionConstants.CP_GENERIC_ERROR, e, e);
      }
    }
  }

  return doc;
}


private String getChildElementValue(Element el,
                                    String tagName)
  throws CPException
{
  String value = null;
  try {

    NodeList  nl = el.getElementsByTagName(tagName);
    if (nl.getLength() > 0) {

      Element elChild = (Element) nl.item(0);
      NodeList nl1 = elChild.getChildNodes();
```

```
        for(int i=0; i<nl1.getLength(); i++) {

          Node nChild = nl1.item(i);
          int  iNodeType = nChild.getNodeType();
          if (iNodeType == Node.TEXT_NODE) {
            value = nChild.getNodeValue();
            break;
          }
        }
      }
    }
    catch (Exception e) {
      throw new CPException(CPExceptionConstants.CP_GENERIC_ERROR, e, e);
    }

    return value;
  }

  private Element getChannelElement(Element el,
                                    String chName)
    throws CPException
  {
    Element elch = null;
    try {

      NodeList nl = el.getElementsByTagName("channel");
      for (int i=0; i<nl.getLength(); i++) {

        Element elTemp = (Element) nl.item(i);
        String  chTitle = getChildElementValue(elTemp, "title");
        if (chTitle.equals(chName)) {
          elch = el;
          break;
        }
      }
    }
    catch (Exception e) {
      throw new CPException(CPExceptionConstants.CP_GENERIC_ERROR, e, e);
    }

    return elch;
  }

  private Date getDate(String sd)
    throws CPException
```

```
{
  Date d = null;
  try {

    if ((sd != null) && (sd.trim().length() != 0)) {
      SimpleDateFormat sdf = new SimpleDateFormat(LAST_BUILD_DATE_FORMAT);
      d = sdf.parse(sd);
    }
  }
  catch (Exception e) {
    throw new CPException(CPExceptionConstants.CP_GENERIC_ERROR, e, e);
  }

  return d;
}

public static void main(String argv[])
  throws Exception
{
  String rssurl = argv[0];
  String  offid = argv[1];

  RSSCPAdaptor rsscp = new RSSCPAdaptor();
  rsscp.init( new oracle.syndicate.server.cp.impl.CPContextImpl());
  rsscp.setRSSURL(rssurl);

  Iterator itOffs = rsscp.buildCatalog();
  while (itOffs.hasNext()) {

    CPOffer cpoff = (CPOffer) itOffs.next();
    System.out.println("id: "+cpoff.getCPOfferID());
    System.out.println("name: "+cpoff.getName());
    System.out.println("desc: "+cpoff.getDescription());
    System.out.println("rh: "+cpoff.getRightsHolder());
    System.out.println("pn: "+cpoff.getProductName());
  }

  CPPackage pkg = rsscp.buildPackage( new CPPackageRequestImpl(null, null,
                                                               offid, null,
                                                               null),
                                      null);
  Iterator itItems = pkg.getPackageItems();
  while (itItems.hasNext()) {

    CPItem cpi = (CPItem) itItems.next();
```

```
      System.out.println("id: "+cpi.getID());
      System.out.println("title: "+cpi.getName());
      System.out.println("actv: "+cpi.getActivation());

      ContentAccessor ca = cpi.getContent();
      InputStream cais = ca.openStream();
      ByteArrayOutputStream baos = new ByteArrayOutputStream();
      IOUtils.pipe(cais, baos);
      System.out.println(baos.toString());

      System.out.println("");
    }
  }
}
```

# Index