Oracle® Fusion Middleware

Developer's Guide for Oracle IRM Server

11*g* Release 1 (11.1.1)

E12326-02

May 2010

ORACLE®

Oracle Fusion Middleware Developer's Guide for Oracle IRM Server, 11*g* Release 1 (11.1.1)

E12326-02

Primary Author: Martin Wykes

Contributing Author: James Leask

Contents

3 Sealed Content Examples

7 Code Samples for Java Applications

Preface

This guide describes how to create code that will perform sealing and related tasks for protecting the content of files under Oracle IRM control.

Audience

This guide is for developers who want to create tools for sealing and unsealing protected content, or who want to adapt existing tools to support content protected by Oracle IRM. Users of this guide need basic familiarity with Oracle IRM, should be competent Java developers familiar with a Java IDE, and should know how to call web services from Java.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible to all users, including users that are disabled. 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. Accessibility standards will continue to evolve over time, and Oracle 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 more information, visit the Oracle Accessibility Program Web site at http://www.oracle.com/accessibility/.

Accessibility of Code Examples in Documentation

Screen readers 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, some screen readers 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 does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Access to Oracle Support for Hearing-Impaired Customers

Oracle customers have access to electronic support through My Oracle Support or by calling Oracle Support at 1.800.223.1711. Hearing-impaired customers in the U.S. who wish to speak to an Oracle Support representative may use a telecommunications relay service (TRS). Information about the TRS is available at http://www.fcc.gov/cgb/consumerfacts/trs.html, and a list of telephone

numbers is available at http://www.fcc.gov/cgb/dro/trsphonebk.html. International hearing-impaired customers should use the TRS at +1.605.224.1837. An Oracle Support engineer will respond to technical issues according to the standard service request process.

Related Documents

For more information, see the following documentation:

- *Oracle Fusion Middleware Administrator's Guide for Oracle IRM Server*

  This guide is also available as the online help for the Oracle IRM Server product.
- *Oracle Fusion Middleware External Users Support Guide for Oracle IRM Desktop*
- *Oracle Fusion Middleware User's Guide for Oracle IRM Desktop*

  This guide is also available as the online help for the Oracle IRM Desktop product.

Conventions

The following text conventions are used in this document:

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

The following conventions are used throughout this guide:

- The notation <*Install_Dir*> is used to refer to the location on your system where Oracle IRM Server is installed.
- Forward slashes (/) are used to separate the directory levels in a path name. This is true when referring to files on a Windows file system or on a UNIX system. A forward slash will always appear after the end of a directory name.

1
Introduction

This guide covers all of the development topics relating to the Oracle IRM J2EE application (Oracle IRM Server).

The guide describes how to use the following:

- **Web services.** The Oracle IRM Web services include operations that allow content to be sealed and unsealed, contexts to be created and rights to be assigned.

  This guide describes the web services that are available for use by external developers. Other web services exist for Oracle IRM that are not authorized for use by external developers.
- **Java APIs.** Sealing and other operations can be performed using the IRM Java API.
- **Status page customization.** On-line Oracle IRM Desktop status pages can be customized allowing an organization to offer a branded or personal experience to end users when working with sealed content.

Where appropriate, sample code is provided.

2

Working with Sealed Content

This section contains the following topics:

2.1 Concepts

This section contains the following topics:

2.1.1 Sealing

Sealing is the process of transforming plaintext content into encrypted and signed content. The sealing process adds metadata, signs this metadata and encrypts the content. The result of this transformation is called sealed content. Sealed content can be opened only with Oracle IRM Desktop, the Oracle IRM client application. Oracle IRM Desktop checks the digital signature, decrypts the content, and maintains the protection of the sealed content while in use. One of the other changes currently made during sealing is to alter the file extension. For example, a sealed HTML document has a stml file extension rather than a html or htm file extension. Oracle IRM Desktop identifies sealed content using these different file extensions.

2.1.1.1 Metadata: The Public Header

The metadata added to sealed content is called the public header. It is a human-readable XML document that appears near the top of the sealed content. The public header is digitally signed so that tampering of sealed content can be detected by Oracle IRM Desktop.

2.1.1.1.1 Classification The public header contains a section called the classification. The classification is used by the Oracle IRM Desktop to determine whether the authenticated user can access the sealed content. Rights are expressed in terms of the classification, for example *John can access all Top Secret classified content*. The

classification also includes information about which server (Oracle IRM Server) to contact for rights, and which cryptography keys were used to seal the content.

Classification Cookie

To allow content to be classified in different ways, the classification contains a section of XML data called the classification cookie. The classification cookie contains data that is used by Oracle IRM Desktop and the Oracle IRM J2EE application to associate rights with content. The data contained in the classification cookie is defined by the classification system.

Context Classified Content

Sealed content that uses the context classification system has a classification cookie that contains a UUID value (to identify a context) and an item code that identifies the document. This allows rights to be expressed either at the context level or for a particular document (for example, *John can access any document sealed to context Top Secret* or *Mary can access the top secret document named secrets.sdoc*).

2.1.1.1.2 Custom Metadata Additional data can be tagged to sealed content using custom metadata. Custom metadata can be added by third party systems that perform sealing. This allows tamper proof metadata to be added to sealed content, which in turn can be extracted by these applications. For example, a content management system could add additional properties to the sealed content, such as the original author or document version.

Oracle IRM Desktop also uses custom metadata when displaying the poster page for sealed movies. When movie content is sealed the poster page can be specified as custom metadata.

2.1.1.1.3 Content Schema Sealed content contains a version number called the content schema. This version number helps the Oracle IRM Desktop determine what features are supported in the sealed content.

2.1.1.1.4 Creation Time Sealed content contains a record of when the content was first sealed and when subsequent edits were made.

2.1.1.1.5 Example Public Header The following XML document is an example of a public header one might see in an HTML document sealed against the Top Secret context.

```
<?xml version="1.0" ?>
<content:PublicHeader xmlns:content="http://xmlns.oracle.com/irm/content">
    <contentDescription>
        <schema>
            <schemaVersion>
                <version>6.0</version>
            </schemaVersion>
        </schema>
        <classification>
            <id>588403f9-9cff-4cce-88e4-e030cc57282a</id>
            <system>
                <uuid>37c8da32-5420-4146-816c-27f63de27250</uuid>
            </system>
            <keySet>
                <uuid>213f8f65-c5d1-4868-9fff-ad156daa2dd6</uuid>
            </keySet>
            <uri>http://irm.example.com/irm_desktop</uri>
            <classifications:ContextCookie
xmlns:classifications="http://xmlns.oracle.com/irm/classifications">
```

```
                    <context>
                        <uuid>588403f9-9cff-4cce-88e4-e030cc57282a</uuid>
                    </context>
                    <itemCode>
                        <value>example.stml</value>
                    </itemCode>
                </classifications:ContextCookie>
                <classificationTime>2008-02-01T13:00:00.000+01:00</classificationTime>
                <labels>
                    <locale>en</locale>
                    <name>Top Secret</name>
                </labels>
            </classification>
            <customData>
            <uuid>2b8cd20a-d4f5-47b6-9097-d12547f2b707</uuid>
        <acme>
<author>John Smith</author>
<version>2</version>
    </acme>
</customData>
            <creationTime>2009-01-01T12:00:00.000+01:00</creationTime>
            <editTime>2009-01-01T12:00:00.000+01:00</editTime>
            <sealedMime>application/vnd.sealedmedia.softseal.html</sealedMime>
            <unsealedSize>2367</unsealedSize>
    </contentDescription>
    <iv>d2hhdCB3aWxsIHByaW50IG91dA==</iv>
    <sessionKey>SGVsbG8gTW9vbiBNb25rZXk=</sessionKey>
    <publicHeaderPeriod>1024</publicHeaderPeriod>
    <encryptedContentBlockSize>16384</encryptedContentBlockSize>
</content:PublicHeader>
```

2.1.1.2 Encrypted Content

The following is an example snippet of the encrypted section of a sealed file.

```
O£´–ä#Dmbg...l]>:z÷ýëËÜžXçÚÔÞôü•›@Óª‹Ý Ž•¤Ð3
Jòo|(0r8Cª3OÁJV'ž™ýZ{÷²V-Š°Âl§o*ÒàY¢ä)è µRTÑ
‹› - î†ê$<óóPVëcÏ¬®þÒ,œ- A-:n«HC"±>œUNµ´®^
#•ƒÃ×¾[Ò@»"C¯V¼@WL¼mÌÀG§ú)ê{Ô=Ya®fýÂÔØ¸oLP
'dúÏ"w)<.1äÊÅb< Ë1•IJñuãŒ¯':^Ÿ.}eTS˜påÕ@J
;¯Y•u,<º¦R¢hZ_-1qeÏD&hú§ŒF+wsè\¡ž'Ë'éðñÚGÃ
```

2.1.2 Unsealing

Unsealing is the process of taking sealed content and extracting the original, plaintext content. Unsealing can be considered the reverse process of sealing. Unsealing is typically used when content no longer requires Oracle IRM protection or when the content needs to be processed by a third party system, for example an application producing a search index for sealed content.

2.1.3 Peeking

Peeking is the process of extracting the classification and custom metadata from sealed content. The process is called peeking because the process examines only the public header of the sealed content, not the encrypted data. Peeking is typically used to identify the classification of the sealed content without opening or viewing the content. Peeking can also check the digital signature: this is called validated peek. Peeking of this form requires the cryptography keys to be available to the caller, which typically means the authenticated user must have rights to open the sealed content.

2.1.4 Resealing

Resealing is the process of saving a sealed file with some modifications. Oracle IRM Desktop allows certain formats, such as Microsoft Office, to be edited in sealed form: the process of saving edits is called resealing.

2.1.5 Reclassification

Reclassifying sealed content is the process of altering the classification of the sealed content. Reclassification usually means re-signing and re-encrypting the content as most classifications have their own set of cryptography keys. Reclassifying is typically used when content changes sensitivity (for example, a top secret document becomes a company confidential document).

2.2 Sealing Server

The sealing server is a J2EE application that runs on an application server and provides sealing, unsealing, peeking, and reclassification services. Operations such as sealing content can be accomplished by uploading a file to the sealing server: the relevant metadata will be added and digitally signed, and the contents encrypted. The resultant sealed content is then returned to the caller. The sealing operations are exposed as web services. The typical use for the sealing server is for integrations that want to seal, unseal or examine sealed content. Oracle IRM Desktop users typically create sealed content on their local machines and would not use the sealing server.

The sealing server exposes two web services: the sealing web service and the desktop web service. The sealing web service allows sealed content to be manipulated (for example, sealing and unsealing content). The desktop web service allows licenses checked out to the sealing server to be queried and checked in. The desktop web service also provides classification details.

2.2.1 Sealing Web Service

The sealing server provides a number of web services operations that allow the following tasks to be performed:

- Sealing unsealed content
- Unsealing sealed content
- Resealing or reclassifying sealed content
- Peeking sealed content metadata

The sealing services WSDL file can be downloaded from the sealing server using the following URL, replacing `irm.example.com` with the host and port name of the sealing server:

```
http://irm.example.com/irm_sealing/sealing_services?wsdl
```

2.2.1.1 Authentication

The sealing services web service calls require authentication. The current release supports HTTP basic authentication. When rights are requested and operations performed on sealed content, they are performed as the authenticated user.

2.2.1.1.1 Setting the Username and Password using JAX-WS There are two main ways to set the username and password with a JAX-WS generated web service proxy. The first approach requires the user name and password to be set using the

`java.net.Authenticator` class. This approach provides a user name and password for all HTTP requests for the running JVM instance.

```
java.net.Authenticator.setDefault(new java.net.Authenticator() {
            @Override
            protected PasswordAuthentication getPasswordAuthentication() {
                return new PasswordAuthentication("username",
"password".toCharArray());
            }
        });
```

The other approach is to set the user name and password directly on the web services port object. This is the approach used in the web service sample code.

```
java.util.Map<String, Object> requestContext =
(javax.xml.ws.BindingProvider)port).getRequestContext();
        requestContext.put(javax.xml.ws.BindingProvider.USERNAME_PROPERTY,
"username");
        requestContext.put(javax.xml.ws.BindingProvider.PASSWORD_PROPERTY,
"password");
```

2.2.1.2 Authorization

The sealing services web service calls are authorized in the same way that Oracle IRM Desktop authorizes sealed content access. The authenticated user must have a valid license that allows an appropriate feature for the specified classification.

- **Sealing** - requires a license that allows the *seal* feature.

  In the context classification system, this means the user has to have a role assigned that allows the *seal* feature.

- **Unsealing** - requires a license that allows the *save unsealed* feature.

  In the context classification system, this means the user has to have a role that has export constraints set as *none*.

- **Resealing** - requires a license that allows the *reseal* feature.

  In the context classification system, this means the user has to have a role that allows the *reseal* feature.

- **Reclassification** - requires a license that allows the *copy to* feature with an appropriate trusted destination in the source classification license and the *seal* feature in the target classification license.

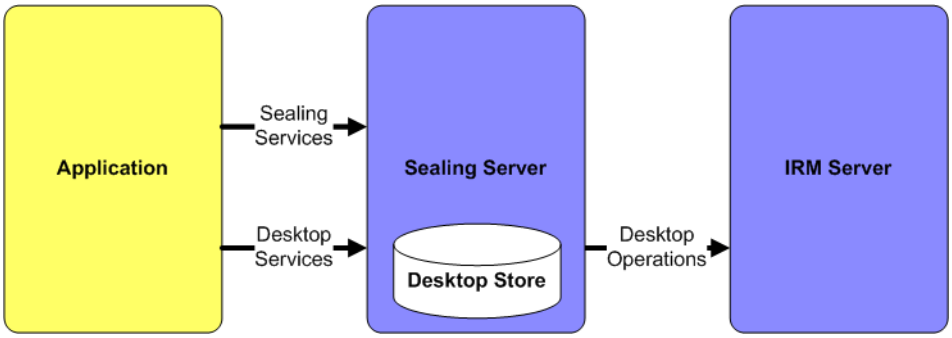
  In the context classification system, this means the user has to have a role that has export constraints set as *trusted* with the target context being a trusted context of the source context, or that the role has export constraints set as *none*.

- **Peeking** - can be performed without having a license (unless the digital signature is checked, in which case the *open* feature is required).

  In the context classification system, this means that the user does not need a role assigned within a context.

If the same user account is used to authenticate with the sealing server and the Oracle IRM Desktop, the user may have to perform a check-in when switching between Oracle IRM Desktop and the sealing server. It is advisable to use a different user account when using the sealing server, to avoid having to check in licenses. When requests are made to the sealing server, the sealing server requests the licenses and cryptography keys for the related classification from the Oracle IRM server. These licenses are checked out to the sealing server for the authenticated user. A record of these keys and rights is stored in memory and, if the license specifies, also on the file

system. These licenses and keys are then used to process the content for this call and subsequent requests relating to the same classification (for the same user). Licenses that expire are re-requested by the sealing server. License rules, such as time constraints, are fully interpreted by the sealing server. License and key details are stored per authenticated user - there is an isolated store of rights and keys for any user that uses the sealing server. This is identical to the way that Oracle IRM Desktop requests licenses and keys. Oracle IRM Desktop also has an offline database in which keys and licenses are stored.



2.2.1.3 MTOM

The sealing server supports MTOM (SOAP Message Transmission Optimization Mechanism). This web service feature allows content to be transmitted as a raw binary attachment to the SOAP message rather than using in-line base 64 encoded data. This not only reduces the amount of data sent to the server, it also allows larger files to be uploaded and downloaded. It is strongly recommended to enable MTOM support when using the sealing web service operations.

For example, when using a JAX-WS client the MTOM feature can be enabled when obtaining the port:

```
service.getPort(SealingServices.class, new javax.xml.ws.soap.MTOMFeature());
```

If MTOM support is not enabled, the uploaded file size will be limited to the available memory in the client JVM.

2.2.2 Desktop Web Service

The sealing server also exposes a number of operations for managing licenses checked out to the sealing server and listing classification details. These operations are similar to the operations a user can perform using Oracle IRM Desktop. When licenses are checked out to the sealing server, the licences cannot be used by Oracle IRM Desktop unless the device count configuration setting has been configured to allow licenses to be used on multiple devices. The desktop services web service allows licenses to be checked in from the sealing server. In the same way that Oracle IRM Desktop can list classification details (contexts) when the user wants to seal content, the sealing server also provides an operation that lists classification details.

A couple of simple diagnostic pages are also made available on the sealing server so you can find out what licenses are checked out to the sealing server, and what classifications are currently available:

```
http://irm.example.com/irm_sealing/licenses
```

```
http://irm.example.com/irm_sealing/classifications
```

The desktop services WSDL file can be downloaded from the sealing server using the following URL, replacing `irm.example.com` with the host and port name of the sealing server.

```
http://irm.example.com/irm_sealing/desktop_services?wsdl
```

2.3 IRM Java API

Applications that want to process sealed content locally rather than sending the content to a server can use the IRM Java API. When using this API the sealed content cryptography is performed in the same process as the calling application. Cryptography keys are shipped from the IRM J2EE application and used in the process using the API.

> **Cryptography Keys:** An application using the IRM Java API has the ability to obtain the sealed content cryptography keys. The IRM Java API stores these keys in the memory of the calling process whilst they are being used. Debugging the IRM Java API, or doing a memory dump could compromise the key material. Do not use the IRM Java API if the environment in which the application is running is insecure or untrusted (for example, a laptop).

The typical use for the IRM Java API is for trusted integrations that want to seal, unseal or examine sealed content where network latency or performance may be an issue.

The API implementation makes use of Java Cryptography Extension (JCE) to perform cryptography operations. If use of AES 256 cryptography is required, the unlimited strength policy JAR files must also be installed. For the latest information about supported JDK/JREs, see the System Requirements and Supported Platforms for Oracle Fusion Middleware page on Oracle Technology Network at http://www.oracle.com/technology/software/products/ias/files/fusion_certification.html.

When using the IRM Java API, the following jar file should be added to the compile time or runtime classpath:

- `irm-api.jar`

This `irm-api.jar` has a manifest entry that adds the following jars files to the classpath. These jar files need to be distributed with the `irm-api.jar` and placed in the same folder as the `irm-api.jar` file.

- `irm-common.jar`
- `irm-engine.jar`
- `irm-client.jar`
- `irm-ws.jar`

These jar files can be obtained from the IRM server installation, under the folder `ECM_ORACLE_HOME/irm/api/lib`.

2.3.1 Runtime Configuration

Configuration settings for the IRM Java API are read from an XML configuration file. The configuration settings tell the IRM Java API where to store the offline cache of licenses, classification and cryptography key information, and the device identity, as

well as where to find the key store containing the cryptography key used to protect sealed content cryptography key material.

```
<?xml version="1.0" encoding="UTF-8" ?>
<core:DesktopConfiguration xmlns:core="http://xmlns.oracle.com/irm/core">
    <storageFolder>/home/irm</storageFolder>
    <device>6f750242-c35e-4992-bb37-e160dba87779</device>
    <application>
<name>sample-application</name>
    </application>
    <keyStore>
        <type>JCEKS</type>
        <path>/home/irm/irm_api.jceks</path>
    </keyStore>
</core:DesktopConfiguration>
```

This XML configuration file can be stored in any location accessible to the process running the IRM Java API. The location is specified by setting the `oracle.irm.client.configuration` system property. If this property is not set, the IRM Java API will throw an exception, informing the caller than the configuration settings have not been specified. Here is an example of a Java program running with the system property set to `/home/irm/irm-configuration.xml`:

```
java -Doracle.irm.client.configuration=/home/irm/irm-configuration.xml MyProgram
```

2.3.1.1 Storage Folder

When the IRM Java API is used to seal, unseal, reseal and reclassify content, license and cryptography key details are requested from the IRM J2EE application for the authenticated user. Depending on the license rules these details may be stored in memory or on the file system and used for future requests to seal, unseal, reseal, or reclassify content. The storage folder is where this data is stored. This data is stored in encrypted form and is locked to the machine that requested the information.

> **Authentication:** Once license and key material has been obtained from the IRM J2EE application, the process using the IRM Java API will be able to process sealed content, regardless of the user used to authenticate the original request. Unlike the sealing server, these details are shared per process rather than per user.

The following two files will be created the first time an operation is performed that needs to store license related details:

- `irm-store/irm-desktop-store.xml`
- `irm-key-sets/irm-key-set-48ff59ab-be7a-4a2a-a093-65286f949909.xml`

  The UUID value `48ff59ab-be7a-4a2a-a093-65286f949909` will vary from installation to installation.

> **Backups:** If the contents of the storage folder are deleted, the next time the IRM Java API saves information to the file system these files will be recreated. There is no need to back up the files generated by the IRM Java API.

2.3.1.2 Device

Applications that perform sealing operations, such as Oracle IRM Desktop, sealing server and the IRM Java API, provide details about the where the operation is being performed. This information is provided in the form of a *device*. Each device has a unique value that is used to identify the location of an Oracle IRM client to the IRM J2EE application. This value is in the form of a UUID and is called the *device UUID*. When using the IRM Java API, a new UUID value should be generated and specified in the configuration file. It is valid to allow logically identical devices to share the same device UUID value. For example, a cluster of sealing services shares the same device UUID, so every member of the cluster can obtain the same licenses and the cluster is seen as one logical location.

The device UUID is used internally by IRM to track the location of licenses. When a license is used, it is typically checked out to a device (depending on the license criteria). This prevents the license being used on multiple devices at the same time. Sharing the same device UUID is however a simple way to relax this restriction when scaling or clustering an application that uses the IRM Java API.

2.3.1.3 Application

The application name is a human-readable label for the application that is processing sealed content. The application name is part of the information provided to the IRM J2EE application in a content-related audit record.

2.3.1.4 Key Store

When the IRM Java API obtains key material from the IRM J2EE application, the key material is encrypted using a key dedicated to the device. To be able to use the key material, the IRM Java API must have access to the appropriate cryptography key so that the key material can be decrypted and used to process sealed content. The key store type and location are specified in the configuration file. Passwords for the key store and key are not stored in the configuration file. Passwords will be prompted for on the command line when the IRM Java API first accesses the key store.

2.3.2 Enabling the IRM Java API

By default, the Oracle IRM J2EE application will only respond to requests from the Oracle IRM Desktop and the sealing server. To use the IRM Java API, additional configuration steps are required to add the application using the IRM Java API as a trusted application. Processing sealed content locally using the IRM Java API requires local access to the content cryptography keys. These keys are requested from the Oracle IRM J2EE application, which will respond with key material together with licenses. These licenses control what operations the IRM Java API is allowed to perform.

Without enabling the IRM Java API, requests to process seal content will result in the following error:

```
oracle.irm.engine.core.desktop.RepudiateException: IRM-01023: The desktop is
repudiated
```

Peeking: If an application is only using the peek operation to extract sealed content metadata, there is no need to set up a trust relationship between the application using the IRM Java API and the IRM J2EE application. Peeking does not use sealed content cryptography keys while peeking content.

2.3.2.1 Generating a Key

The first step in enabling the IRM Java API to use an IRM J2EE application is to generate a new AES symmetric key or RSA asymmetric key pair. This can be done with the Java `keytool` command line tool. This AES symmetric key or RSA asymmetric key pair are used as a trust mechanism. The IRM J2EE application will encrypt data using a key, and the IRM Java API will decrypt data using the corresponding key.

> **Key Algorithm Choice:** On AIX platforms using the IBMJCE cryptography provider, the AES key wrap algorithm is not supported, so a RSA 2048 bit trust key must be used. If the IRM J2EE application is using a JKS key store type, the IRM Java API is also limited to using a RSA key. In all other cases, generate an AES key which is at least the same size as the sealed content key sizes. For example, when using the AES128 cryptography schema, generate a 128 or 256 bit AES key for the IRM Java API.

The following is an example of generating a 128 bit AES key and storing it in a JCEKS file based key store. The key alias should be the device UUID specified in the IRM Java API configuration. In the examples shown, the device UUID is `b2e1ea48-cc3e-44bc-ad9b-a214c62fd410` and the key store is called `irm_api.jks/irm_api.jceks`:

```
keytool -keystore irm_api.jceks -storetype JCEKS -genseckey -alias
b2e1ea48-cc3e-44bc-ad9b-a214c62fd410 -keysize 128 -keyalg AES
```

The following is an example of generating a 2048 bit RSA key and storing it in a JKS file based key store:

```
keytool -keystore irm_api.jks -keysize 2048 -genkeypair -alias
b2e1ea48-cc3e-44bc-ad9b-a214c62fd410 -keyalg RSA
```

These commands all create the key store, generate a key, and prompt the user for a key store and key password. This key store is the one that should be used in the IRM Java API configuration.

> **Certificate:** When generating the RSA key, `keytool` will prompt for certificate details. The certificate details are not used by the IRM Java API or IRM J2EE application, so enter blank values for the certificate questions.

2.3.2.2 Adding the Key to the Server

The symmetric key or asymmetric public key generated with `keytool` must be added to the IRM J2EE application key store, using the device UUID value as the key alias. The IRM J2EE application then uses this key to encrypt key material before sending it to the IRM Java API. Without the key being added to the IRM J2EE application key store, the IRM Java API will not be able to process sealed content.

> **Location:** The IRM J2EE application key store is typically located under the folder `DOMAIN_HOME/config/fwmconfig`.

2.3.2.3 AES Keys

If the IRM Java API key is an AES key then the `keytool -importkeystore` feature can be used to import the key into the server key store. In this example, the IRM Java API key store is called `irm_api.jceks` and the IRM J2EE application key store is called `irm.jceks`:

```
keytool -importkeystore
  -srckeystore irm_api.jceks
  -srcstoretype JCEKS
  -srcalias b2e1ea48-cc3e-44bc-ad9b-a214c62fd410
  -destkeystore irm.jceks
  -deststoretype JCEKS
  -destalias b2e1ea48-cc3e-44bc-ad9b-a214c62fd410
```

When specifying the key store file name, the key store path may also be required, depending on whether the `keytool` command is run in the same folder as the key store file.

2.3.2.4 RSA Keys

The first step is to export the public key part of the asymmetric key pair from the IRM Java API key store. In this example, the IRM Java API key store is called `irm_api.jks` and the IRM J2EE application key store is called `irm.jks`.

```
keytool -exportcert -keystore irm_api.jks -alias
b2e1ea48-cc3e-44bc-ad9b-a214c62fd410 -file
b2e1ea48-cc3e-44bc-ad9b-a214c62fd410.cer
```

The next step is to import this public key into the IRM J2EE application key store.

```
keytool -importcert -alias b2e1ea48-cc3e-44bc-ad9b-a214c62fd410 -file
b2e1ea48-cc3e-44bc-ad9b-a214c62fd410.cer -keystore irm.jks
```

When specifying the key store file name, the key store path may also be required, depending on whether the `keytool` command is run in the same folder as the key store file.

2.3.2.5 Setting the Password

The IRM J2EE application will need to read the imported key and so will need to know the key password for symmetric keys. For asymmetric there is no password associated with the public key, so this step can be skipped. The key password for the IRM J2EE application is stored in the Fusion Middleware Credential Store Framework (CSF). The WLST tool is used to add new passwords. A WLST shell should be launched from `ECM_ORACLE_HOME/common/bin`. The credential is added using the `createCred` command. The example below shows connecting to an admin server and setting the password for a key store called `irm.jceks` to the value password for the key with alias `b2e1ea48-cc3e-44bc-ad9b-a214c62fd410`.

```
connect("weblogic", "password", "t3://adminServerHost:adminServerPort")
createCred("IRM", "key:irm.jceks:b2e1ea48-cc3e-44bc-ad9b-a214c62fd410", "dummy",
"password")
```

In this example, the key alias is `b2e1ea48-cc3e-44bc-ad9b-a214c62fd410` and the key store file name is `irm.jceks`. Both these values should be altered to reflect the device UUID used and the actual IRM J2EE application key store file name. As keys do not require a user name, the value `dummy` is used in place of a user name.

2.3.3 Authentication

A sealed content operation that ends up requesting licenses and keys may involve a request to the IRM J2EE application, which in turn will require authentication. A remote call is made from the IRM Java API if the local memory and file cache of licenses and keys cannot satisfy the request. The peek operation does not require access to licenses and keys, and will never cause a request to the IRM J2EE application.

2.3.3.1 Using a `java.net.Authenticator`

A `java.net.Authenticator` can be used to specify the username and password. Any HTTP connection that requires authentication will call back to the authenticator to request credentials.

```
java.net.Authenticator.setDefault(
    new java.net.Authenticator() {
        @Override
        protected PasswordAuthentication getPasswordAuthentication() {
            return new PasswordAuthentication("username",
"password".toCharArray());
        }
    });
```

This code sample uses hard-coded passwords and user names. In a production system, these details should be retrieved from a secure location or a user prompt.

2.3.4 Advanced Topics

The IRM Java API provides a plug-in mechanism that allows code to be augmented into internal processes. Two such processes are the code that provides the key store and key password for the wrapping keys, and the code that reads configuration settings. Both processes can be changed to use programmer-provided logic instead of using the default IRM Java API logic.

2.3.4.1 Supplying Configuration Settings in Code

The configuration settings can be provided using code rather than using an XML configuration file. If code is used, there is no need to specify a location for the XML configuration file as a system property. The sample code below shows how configuration settings can be provided using code. The settings are hard-coded and are identical to the XML sample configuration file. In a production system this code could read the settings from an alternative location rather than relying on an externally created XML file.

```
import static
oracle.irm.engine.core.desktop.DesktopApplicationFactory.createDesktopApplication;
import static
oracle.irm.engine.content.store.KeyStoreSettingsFactory.createKeyStoreSettings;

import oracle.irm.engine.core.desktop.DesktopApplication;
import oracle.irm.engine.content.store.KeyStoreSettings;
import oracle.irm.engine.core.config.DesktopConfigurationSlice;
import oracle.irm.engine.util.Aspect;
import oracle.irm.engine.util.Purpose;
import java.io.File;
import java.util.UUID;

@Aspect(Purpose.IMPLEMENTOR)
public final class DesktopConfigurationDemo extends DesktopConfigurationSlice {
```

```
    @Override
    public void construct() {

        UUID deviceUuid = UUID.fromString("6f750242-c35e-4992-bb37-e160dba87779");

        File storageFolder = new File("/home/irm");

        KeyStoreSettings keyStore= createKeyStoreSettings("JCEKS", new
File("/home/irm/irm_api.jceks"));

        DesktopApplication application = createDesktopApplication("demo");

        vthis.construct(
            storageFolder,
            deviceUuid,
            application,
            keyStore);
    }
}
```

2.3.4.2 Specifying Key and Key Store Passwords in Code

The following code sample shows how to specify key and key store passwords. In a production integration, these passwords should be obtained using an appropriate mechanism and passed into the construct method rather than using hard-coded values. The key parameter provides details of the key store and key being requested. This key parameter can be used if the key store path or key alias name are needed to choose an appropriate set of passwords.

```
import oracle.irm.engine.content.key.WrappingKeySlice;
import oracle.irm.engine.content.store.KeySettings;
import oracle.irm.engine.content.store.KeyStoreAccessException;
import oracle.irm.engine.content.store.UnknownKeyException;
import oracle.irm.engine.util.Aspect;
import oracle.irm.engine.util.Purpose;

@Aspect(Purpose.IMPLEMENTOR)
public final class WrappingKeyDemo extends WrappingKeySlice {

    @Override
    public void construct(KeySettings key) throws KeyStoreAccessException,
UnknownKeyException {

        vthis.construct(
            key,
            "password".toCharArray(),
            "password".toCharArray());
    }
}
```

2.3.4.3 Activating Code Plug-ins

To activate code plug-ins, the XML file `META-INF/oracle_irm_deployment.xml` needs to be added to the classpath. The easiest way to do this is to package this file in the same jar file as the plug-in code and add the jar file to the end of the classpath. This sample file `oracle_irm_deployment.xml` tells the API to add in the `WrappingKeyDemo` and `DesktopConfigurationDemo` classes into the internal logic processes.

```
<system:Deployment xmlns:system="http://xmlns.oracle.com/irm/system">
    <components>
```

```
        <name>demo</name>
        <plugins>
            <interfaceClass>oracle.irm.engine.content.key.WrappingKey</interfaceClass>

<resourcePath>/oracle/irm/engine/content/key/aspects/classes/WrappingKey.xml</resourcePath>
            <slices>
                <implementationClass>WrappingKeyDemo</implementationClass>
            </slices>
        </plugins>
        <plugins>
            <interfaceClass>oracle.irm.engine.core.config.DesktopConfiguration</interfaceClass>

<resourcePath>/oracle/irm/engine/core/config/aspects/classes/DesktopConfiguration.xml</resourcePath
>
            <slices>
                <implementationClass>DesktopConfigurationDemo</implementationClass>
            </slices>
        </plugins>
    </components>
</system:Deployment>
```

When adding your own classes, change the contents of the `implementationClass` element to provide the package and class name of the class created. The name element can be used to give the plug-in a unique name: this could be the name of the company or application providing the plug-in logic. Also ensure the classes that are created are added to the classpath.

3

Sealed Content Examples

This section contains the following topics:

3.1 Finding File Extensions

When sealing content, it is useful to be able to look up the file extension that Oracle IRM Desktop uses. The content operations described later in this section provide useful operations for obtaining file extension information, such as looking up sealed file extensions.

3.1.1 File Extensions

Sealed content uses file extensions that differ from the ones used for unsealed files. For example, a PDF file has the file extension .pdf, whereas a sealed PDF file has the file extension .spdf. The sealed file extension allows Oracle IRM Desktop to identify what file format the sealed content is, and display appropriate sealed file icons. The table below shows some example file extensions and the corresponding sealed file extension.

Table 3–1 Example file formats and extensions

File format	File extension	Sealed file extension
DOC	doc	sdoc
PPT	ppt	sppt
HTML	html, htm	stml
PDF	pdf	spdf
GIF	gif	sgif

3.1.2 MIME Types

Sealed content includes a MIME type in the sealed content metadata. This MIME type is used by Oracle IRM Desktop to identify the format of the unsealed content. The

MIME type is an alternative way of detecting the file format when the content is not stored in a file (for example, a stream of data downloaded from a HTTP server). Unsealed MIME types vary and there are many examples where a single file format has more than one MIME type. For this reason the sealed content contains a sealed MIME type rather than the unsealed content MIME type. For example, a PDF file has the MIME type `application/pdf`, whereas a sealed PDF file will have a MIME type of `application/vnd.sealedmedia.softseal.pdf` added to the metadata.

Opening up sealed content in an editor will show that a sealed MIME type is added to the metadata. For example, the public header of a sealed PDF file:

```
<?xml version="1.0" ?>
<content:PublicHeader xmlns:content="http://xmlns.oracle.com/irm/content"
xmlns:classifications="http://xmlns.oracle.com/irm/classifications">
    <contentDescription>
...
<sealedMime>application/vnd.sealedmedia.softseal.pdf</sealedMime>
...
    </contentDescription>
</content:PublicHeader>
```

3.1.3 Using the Sealing Server

The sealing server provides a web service that can be used to query file and MIME type information. Sealed content file format information is immutable, so consider retrieving this information once and using a local copy when processing sealed files. This will avoid potentially expensive remote calls to a sealing server.

3.1.3.1 Finding the Corresponding Sealed File Name

When sealing a file a common scenario is to create the corresponding sealed file next to the original. The `getSealedFileName` operation takes a path and file name or just a file name and provides the equivalent sealed file name.

```
ContentTypeOperations contentTypeOperations = new
ContentTypeOperationsService().getContentTypeOperations();

String results =
contentTypeOperations.getSealedFileName("/usr/home/john/sample.html");
```

In the example above the results of calling the method would be `"/usr/home/john/sample.stml"`.

3.1.3.2 Obtaining Content Type Information

A content type object contains all the file type information for content that can be sealed. The content type specifies the file extension(s), its sealed file extension, and the associated MIME types. Content type objects can be obtained using the file extension or the MIME type of the sealed or unsealed content.

```
ContentTypeOperations contentTypeOperations = new
ContentTypeOperationsService().getContentTypeOperations();

ContentType results = contentTypeOperations.getContentTypeFromExtension("pdf");
```

3.1.4 Using the IRM Java API

The IRM Java API supplies a set of methods that can be used to query MIME type and file extension details when working with sealed content.

3.1.4.1 Finding the Corresponding Sealed File Name

Given an unsealed file name, the corresponding sealed file name can be found using the `getSealedFileName` operation. This method is useful when sealing content and deciding what the output file name should be.

```
import static
oracle.irm.engine.content.type.ContentTypeOperationsInstance.getSealedFileName;
..
String results = getSealedFileName("/usr/home/john/sample.html")
```

3.1.4.2 Obtaining Content Type Information

Given an unsealed or sealed file name, the MIME type and sealed file extension details can be found using the `getContentTypeFromExtension` method.

```
import static
oracle.irm.engine.content.type.ContentTypeOperationsInstance.getContentTypeFromExt
ension;
import oracle.irm.engine.content.type.ContentType;
..
ContentType results = getContentTypeFromExtension("pdf")
```

3.2 Sealing

This section contains the following topics:

- Sealing Using the Sealing Server
- Sealing Using the IRM Java API

3.2.1 Sealing Using the Sealing Server

The sealing server supports sealing. Content is uploaded to the sealing server, encrypted and signed, and the sealed content returned to the caller.

3.2.1.1 Uploading Content

For JAX-WS generated web service proxies the content is provided as a `javax.activation.DataHandler` parameter. Using a data handler allows the web service stack to stream the binary content to the server without having to load the complete file into memory.

```
javax.activation.DataHandler input = new javax.activation.DataHandler(new
FileDataSource("example.html"));
```

The data source does not have to be a file.

3.2.1.2 Calling `seal`

A call to the `seal` method requires the unsealed data (in the form of a `DataHandler`), the MIME type of the unsealed or sealed content (either is fine) and the sealing options. The sealing options contain the classification details, custom metadata, and a few other attributes, such as the time the sealed file was created.

```
SealingServices sealingServices = new
SealingServicesService().getSealingServices(new javax.xml.ws.soap.MTOMFeature());

DataHandler results = sealingServices.seal(input, "txt/html", options);
```

It is important to enable the MTOM web service feature. This ensures the sealed content is uploaded to the server in the most optimal form. It also avoids `java.lang.OutOfMemoryException` exceptions if the uploaded file is large.

To call the seal operation, the authenticated user needs rights that allow the seal feature for the specified classification.

3.2.1.3 MIME Type

The seal method requires the MIME type of the unsealed or sealed content to be specified, for example:

- For HTML content either the `txt/html` or `application/vnd.sealed.txt` MIME types can be used.
- For text content either the `txt/plain` or `application/vnd.sealedmedia.softseal.html` MIME types can be used.

For more information about how to obtain sealed content MIME types and what MIME types are supported, see "File Extensions" on page 3-1.

3.2.1.4 Sealing Options

The sealing options contain the classification, custom metadata and settings that affect how the content is encrypted. The classification is the most important part of the sealed content metadata. The classification contains the opaque XML document called the classification cookie. The classification cookie is the data used by Oracle IRM Desktop and the Oracle IRM J2EE application when associating rights with content. The classification cookie XML structure is defined by the classification system of the sealed content. The context classification system, for example, has an XML structure that includes a UUID to identify the context and a value called the item code which can be used to identify an individual document. The following is an example context cookie that might appear in sealed content:

```
<?xml version="1.0" ?>
<classifications:ContextCookie
xmlns:classifications="http://xmlns.oracle.com/irm/classifications">
    <context>
        <uuid>588403f9-9cff-4cce-88e4-e030cc57282a</uuid>
    </context>
    <itemCode>
        <value>sample.sdoc</value>
        <time>2007-05-10T12:00:00.000+00:00</time>
    </itemCode>
</classifications:ContextCookie>
```

Rights for the context classification system are expressed using this information, for example:

John can access all documents with a context UUID of `f3cd57c1-f495-48aa-b008-f23afa4d6b07`

or:

Mary can access documents with a context UUID of `f3cd57c1-f495-48aa-b008-f23afa4d6b07` *and an item code value of* `plan001.sdoc` *or* `plan002.sdoc`

The classification is mandatory and must be specified in the sealing options. The other sealing option properties are optional.

```
SealingOptions options = new SealingOptions();
```

```
options.setClassification(classification);
```

3.2.1.4.1 Classification ID The classification ID is a simple string value that is used to uniquely identify the classification. The contents and format of the classification ID differ depending on what classification system is used. The classification ID is used to match classification details with master classification details stored on the server. During the seal operation, if the classification labels and key set are not specified the sealing server looks up the master classification definition by classification ID and uses the labels and key set defined on the master classification.

```
classification.setId("a4905cd7-7405-469e-b72c-78d11e959b3a");
```

The classification ID value for the context classification system should be set as the context UUID value. If this value is not set correctly, labels and key set details cannot be automatically set.

3.2.1.4.2 Classification System A classification must specify what classification system is being used to seal the content. A classification system is identified with a UUID value.

```
ClassificationSystemRef system = new ClassificationSystemRef();
system.setUuid("37c8da32-5420-4146-816c-27f63de27250");

classification.setSystem(system);
```

The classification system defines what value should be used as the classification ID, as well as what XML data should be set in the classification cookie. When sealed content is opened in Oracle IRM Desktop this information is sent to the Oracle IRM J2EE application. The Oracle IRM J2EE application then uses the classification system and classification cookie data to determine how rights are obtained for the authenticated user.

The UUID value for the context classification system is `37c8da32-5420-4146-816c-27f63de27250`. This value is immutable and will never change.

3.2.1.4.3 Key Set When content is sealed, the cryptography keys used to encrypt and sign the content are specified using a key set. This value should be set to `null`, and is provided for future feature enhancements.

```
classification.setKeySet(null);
```

3.2.1.4.4 Server When sealed content is opened or created, the rights to open or seal the content must be obtained from an IRM server. A classification has a URI property which must be set to the URI of an IRM server that will provide the licenses and cryptography keys needed to open or seal content for the classification.

```
classification.setUri("https://irm.example.com/irm_desktop");
```

It is important that this value is the same as the the "Server URL" property configured on the General Settings page of the Oracle IRM Server Control Console (the Oracle IRM pages of the Oracle Enterprise Manager Fusion Middleware Control Console).

3.2.1.4.5 Classification Time Rights to access sealed content can include time constraints. One such constraint can be based on the classification time, for example:

allow John to access any sealed content up to one month after the classification time

or:

allow Mary to access any content classified in 2008

The classification time can be set during the sealing process:

```
classification.setClassificationTime(new java.util.Date());
```

If the classification time is not specified, it defaults to the current time by the sealing server.

```
classification.setClassificationTime(null);
```

In the 10g Oracle IRM release the classification time was called the publication time.

3.2.1.4.6 Labels A classification can contain a set of human-readable strings called *labels*. The classification labels are used by Oracle IRM Desktop to show the user classification details, for example informing the user that a document is sealed to the Top Secret classification. If no labels are specified for the classification provided to the seal operation, the sealing server will attempt to fill in the labels from the master classification definition. To allow for multi-language support, labels have a locale property. If the classification can be translated into multiple languages, multiple labels can be provided, each one specifying the appropriate locale (for example, `en` for English or `zh-CN` for traditional Chinese - see Locale Codes). Oracle IRM Desktop picks the most appropriate label based on the installed Oracle IRM Desktop language.

For the context classification system, the context labels defined on the Oracle IRM Server Management Console are the ones that are sealed into content.

Empty labels are specified by setting an empty set of labels using `null`.

```
classification.setLabels(null);
```

Labels can also be provided during the sealing process: these override any master classification definition.

```
Label label = new Label();
label.setLocale("en");
label.setName("Top Secret");
label.setDescription("Top Secret - this is a top secret document");

classification.setLabels(new Label[] {label});
```

3.2.1.4.7 Classification Cookie The classification cookie is defined by the classification XML schema as an `<any>` element, that is, an XML element of any form. The structure of this XML document is defined by the classification system being used. Depending on the web service proxy generator used, the cookie XML is typically provided as a `org.w3c.dom.Element` or `Object`. The following code snippet shows a classification cookie for the context classification system. The cookie XML document is created using standard Java document object model (DOM) APIs. It does not matter how the XML document object is created: loaded from a file, created from a string, loaded in from a stream, etc. This example shows the cookie XML being created from a string.

```
String xml =
    "<?xml version=\"1.0\" encoding=\"UTF-8\"?>" +
    "<classifications:ContextCookie
xmlns:classifications=\"http://xmlns.oracle.com/irm/classifications\">" +
    "    <context>" +
    "        <uuid>a4905cd7-7405-469e-b72c-78d11e959b3a</uuid>" +
    "    </context>" +
    "    <itemCode>" +
    "        <value>sample.shtml</value>" +
```

```
    "        <time>2007-05-10T12:00:00.000+00:00</time>" +
    "    </itemCode>" +
    "</classifications:ContextCookie>";

java.io.ByteArrayInputStream stream = new
java.io.ByteArrayInputStream(xml.getBytes("utf-8"));

javax.xml.parsers.DocumentBuilderFactory documentBuilderFactory =
javax.xml.parsers.DocumentBuilderFactory.newInstance();

// As the context classification cookie uses namespaces, ensure these are
maintained on parsing the XML
documentBuilderFactory.setNamespaceAware(true);

javax.xml.parsers.DocumentBuilder documentBuilder =
documentBuilderFactory.newDocumentBuilder();

org.w3c.dom.Document document = documentBuilder.parse(stream);
```

3.2.1.4.8 Providing Custom Metadata Custom metadata can be specified during the sealing process. Custom metadata is an optional property on the `SealingOptions`. Custom metadata is added as an XML element together with a UUID value that can be used to identify the custom data when peeking the sealed content.

```
Element element = document.createElement("SampleCustomData");
element.setTextContent("Some example custom data provided as an XML element
containing this text");

CustomData data = new CustomData();

// UUID identifies the custom data, this example uses a fixed example UUID value
data.setUuid("7f79d1e8-fc07-464c-8477-834951e07060");

// Custom data is XML document
data.setData(element);

// Set on the options before sealing
options.setCustomData(new CustomData[] {data});
```

3.2.1.4.9 Sealed Movie Poster Page A poster page is the image shown before a sealed movie is started. Oracle IRM Desktop loads the optional poster page from the custom metadata section of the public header. A poster page must be a JPEG or GIF image. The image is provided in the custom data as base 64 encoded data together with the file type.

```
// UUID for poster page image
CustomData image = new CustomData();
image.setUuid("6f2c8fba-a2cb-4493-8861-45e5cbda1bac");

// Custom data is base 64 encoded data for image
Element imageElement = document.createElement("item");
imageElement.setTextContent("R0lGODlhhQASAPcAAP.....XfNIQAAAOw==");
image.setData(imageElement);

// UUID for poster page file type - either 'gif' or 'jpeg'
CustomData fileType = new CustomData();
fileType.setUuid("38663feb-5df9-4c14-bd75-b557b6dfea18");

// Custom data is XML document
Element imageElement = document.createElement("item");
```

```
imageElement.setTextContent("gif");
image.setData(imageElement);

// Set on the options before sealing
options.setCustomData(new CustomData[] {image, fileType});
```

3.2.2 Sealing Using the IRM Java API

Sealing is the process of encrypting plaintext content and adding signed metadata to that content. This metadata includes the classification details and any custom metadata supplied during the sealing process. Sealing can also be performed by using Oracle IRM Desktop. A call to the seal requires the unsealed data in the form of a source, the MIME type of the unsealed or sealed content (either is acceptable), and the sealing options. The sealing options contain the classification details, custom metadata, and a few other attributes, such as the time the sealed file was created.

```
import static oracle.irm.engine.content.sealing.SealingOperationsInstance.seal;
import static oracle.irm.engine.content.sealing.FileSource.createFileSource;
import oracle.irm.engine.content.sealing.Source;
import oracle.irm.engine.content.sealing.SealingOptions;
...
// Create the unsealed content source
Source source = createFileSource("unsealed.html");

// Sealed file written out to a file
OutputStream output = new FileOutputStream("sealed.stml");

// Sealing options...more on this later
SealingOptions options = ...

// Seal the content
seal(source, output, options);
```

To call the seal operation, the authenticated user needs rights that allow the seal feature for the specified classification.

3.2.2.1 Unsealed Content Source

The source parameter provides the sealing operation with a stream to the unsealed data, the MIME type of the content, and the size of the unsealed content (if available). The MIME type is important because Oracle IRM Desktop uses the MIME type to determine how to render the content. The IRM Java API provides a number of sources: the main ones are listed here.

`FileSource`

The file source provides can be used to seal a file. Internally, the file source provides a `FileInputStream` stream to the sealing code. The MIME type is inferred from the file extension.

```
import static oracle.irm.engine.content.sealing.FileSource.createFileSource;
import oracle.irm.engine.content.sealing.Source;
...
Source source = createFileSource("unsealed.html");
```

`URLSource`

The URL source provides content downloaded from a HTTP or FTP server. Internally, the URL source opens a connection using a `java.net.URLConnection` and provides

a stream to downloaded content. The MIME type is inferred from the content-type header, or if that header is not present, a file extension on the URL path.

```
import static oracle.irm.engine.content.sealing.URLSource.createURLSource;
import oracle.irm.engine.content.sealing.Source;
...
java.net.URL url = new java.net.URL("http://irm.example.com/sample.html");
Source source = createURLSource(url);
```

`StreamSource`

The stream source allows the programmer to supply both the input stream and MIME type. This option gives the programmer control over the type of input stream and MIME type used during sealing.

```
import static oracle.irm.engine.content.sealing.StreamSource.createStreamSource;
import oracle.irm.engine.content.sealing.Source;
...
java.io.InputStream myStream = new MyInputStreamStream();
Source source = createStreamSource(myStream, "text/html");
```

The MIME type can be either the unsealed or sealed content MIME type. For example:

- For HTML content, either the `txt/html` or `application/vnd.sealed.txt` MIME types can be used.
- For text content, either the `txt/plain` or `application/vnd.sealedmedia.softseal.html` MIME types can be used.

3.2.2.2 Sealing Options

The sealing options contain the classification, custom metadata, and settings that affect how the content is encrypted. The classification is the most important part of the sealed content metadata. The classification contains the opaque XML document called the classification cookie. The classification cookie is the data used by Oracle IRM Desktop and the Oracle IRM J2EE application when associating rights with content. The classification cookie XML structure is defined by the classification system of the sealed content. The context classification system, for example, has an XML structure that includes a UUID to identify the context and a value called the item code which can be used to identify an individual document. An example context cookie that might appear in sealed content:

```
<?xml version="1.0" ?>
All rights reserved. -->
<classifications:ContextCookie
xmlns:classifications="http://xmlns.oracle.com/irm/classifications">
    <context>
        <uuid>588403f9-9cff-4cce-88e4-e030cc57282a</uuid>
    </context>
    <itemCode>
        <value>sample.sdoc</value>
        <time>2007-05-10T12:00:00.000+00:00</time>
    </itemCode>
</classifications:ContextCookie>
```

Rights for the context classification system are expressed using this information, for example:

John can access all documents with a context UUID of `f3cd57c1-f495-48aa-b008-f23afa4d6b07`

or:

Mary can access documents with a context UUID of `f3cd57c1-f495-48aa-b008-f23afa4d6b07` and an item code value of `plan001.sdoc` or `plan002.sdoc`

3.2.2.2.1 Classification The classification is mandatory and must be specified in the sealing options. The other sealing option properties are optional.

```
import static
oracle.irm.engine.content.sealing.SealingOptionsFactory.createSealingOptions;
import static
oracle.irm.engine.core.classification.ClassificationFactory.createClassification;
import oracle.irm.engine.content.sealing.SealingOptions;
import oracle.irm.engine.core.classification.Classification;
...
// Create the classification...parameter details are described below
Classification classification = createClassification(
    id,
    system,
    keySet,
    uri,
    classificationTime,
    labels,
    cookie);

// Create sealing options
SealingOptions options = new createSealingOptions(classification);
```

3.2.2.2.2 Classification ID The classification ID is a simple string value that is used to uniquely identify the classification. The contents and format of the classification ID differ depending on what classification system is used. The classification ID is used to match classification details with master classification details stored on the server. During the seal operation, if the classification labels and key set are not specified the IRM Java API looks up the master classification definition by classification ID and uses the labels and key set defined on the master classification.

> **Note:** The classification ID value for the context classification system should be set as the context UUID value. If this value is not set correctly, labels and key set details cannot be automatically set.

Classification System

A classification must specify what classification system is being used to seal the content. The classification system defines what value should be used as the classification ID, as well as what XML data should be set in the classification cookie.

A classification system is identified with a UUID value.

```
import static
oracle.irm.engine.core.classification.ClassificationSystemFactory.createClassifica
tionSystem;
import oracle.irm.engine.core.classification.ClassificationSystem;
...
java.util.UUID uuid =
java.util.UUID.fromString("37c8da32-5420-4146-816c-27f63de27250");
ClassificationSystem system = createClassificationSystem(uuid);
```

When sealed content is opened in Oracle IRM Desktop this information is sent to the Oracle IRM J2EE application. The Oracle IRM J2EE application then uses the

classification system and classification cookie data to determine how rights are obtained for the authenticated user.

> **Note:** The IRM Java API provides the constant `oracle.irm.engine.classifications.context.ContextConstants.CONTEXT_CLASSIFICATION_SYSTEM` which can be used as the system parameter when creating a context}.

Key Set

When content is sealed, the cryptography keys used to encrypt and sign the content are specified using a key set. If there is a selection of key sets available to a classification, a key set can be explicitly set in the classification. This would, for example, allow the caller of the seal method to select a stronger encryption algorithm based on the type of content being sealed. The key set specified must be one the classification allows.

```
import static oracle.irm.engine.content.key.KeySetFactory.createKeySet;
import oracle.irm.engine.content.key.KeySet;
...
java.util.UUID uuid =
java.util.UUID.fromString("52546b5c-a273-46c0-b990-e3538700182e");
KeySet keySet = createKeySet(uuid);
```

However, the most common way to specify which key set to use during the seal process is to let the system automatically select a key set. This is done by using a null key set value. The IRM Java API will then select the most recently generated key set that this classification allows.

```
KeySet keySet = null;
```

> **Note:** When sealing content to the context classification system, the most common approach is to specify null in the key set property and let the IRM Java API select the most up to date key set available for the context.

Server

When sealed content is opened or created, the rights to open or seal the content must be obtained from an IRM server. A classification has a URI property which must be set to the URI of an IRM server that will provide the licenses and cryptography keys needed to open or seal content for the classification.

```
java.net.URI uri = java.net.URI.create("https://irm.example.com/irm_desktop");
```

> **Value:** It is important that this value be the same as the URI value configured on the IRM server as the value sealed into content. The IRM Java API and IRM server use the server URI value as an internal way of indexing and referencing information. If this value is incorrect (for example, because it is using an IP address) the IRM Java API will not authorize the operation.

Classification Time

Rights to access sealed content can include time constraints. One such constraint can be based on the classification time, for example *Allow John to access any sealed content up*

to one month after the classification time or *Allow Mary to access any content classified in 2008*. The classification time can be set during the sealing process.

```
java.util.Date classificationTime = new java.util.Date();
```

If the classification time is not specified, it defaults to the current time by the IRM Java API.

```
java.util.Date classificationTime = null;
```

> **Note:** In the 10g Oracle IRM release, the classification time was called the publication time.

Labels

A classification can contain a set of human readable strings called labels. The classification labels are used by Oracle IRM Desktop to show the user classification details, for example informing the user that a document is sealed to the Top Secret classification. If no labels are provided on the classification provided to the seal operation, the IRM Java API will attempt to fill in the labels from the master classification definition. To allow for multi-language support, labels have a locale property. If the classification can be translated into multiple languages, multiple labels can be provided, each one specifying the appropriate locale, for example `en` for English or `zh-CN` for traditional Chinese. Oracle IRM Desktop picks the most appropriate label based on the installed Oracle IRM Desktop language.

> **Context Classification System:** For the context classification system, the context labels defined on the Oracle IRM Server Management Console are the ones that are sealed into content.

Empty labels are specified by setting an empty set of labels using null.

```
import oracle.irm.engine.core.general.Label;
...
java.util.Collection<Label> labels = null;
```

Labels can also be provided during the sealing process: these override any master classification definition.

```
import static oracle.irm.engine.core.general.LabelFactory.createLabel;
import static oracle.irm.engine.core.general.LabelCollectionFactory.createLabels;
import oracle.irm.engine.core.general.Label;
...
// Create the label with an English locale, a name and description
Label label1 = createLabel("en", "Top Secret", "Top Secret - this is a top secret
document");
Label label2 = createLabel("fr", "Top Secret", "Top secret - c'est un document
extrêmement secret");

// Create the labels
java.util.Collection<Label> labels = createLabels(label1, label2);
```

Classification Cookie

The classification cookie is defined by the classification XML schema as an `any` element; a XML element of any type. The structure of this XML document is defined by the classification system being used. When sealing to the context classification

system the IRM Java API provides a set of objects that can be used instead of providing raw XML documents.

```
import static
oracle.irm.engine.classifications.context.ContextCookieFactory.createContextCookie
;
import static
oracle.irm.engine.classifications.context.ContextFactory.createContext;
import static
oracle.irm.engine.classifications.item.ItemCodeFactory.createItemCode;
import oracle.irm.engine.classifications.context.Context;
import oracle.irm.engine.classifications.context.ContextCookie;
import oracle.irm.engine.classifications.item.ItemCode;
...
// Create an item code, this identifies the sealed content within the context
ItemCode itemCode = createItemCode("sample.shtml");

// Create a context, this is identified with a UUID value
java.util.UUID uuid =
java.util.UUID.fromString("a4905cd7-7405-469e-b72c-78d11e959b3a");
Context context = createContext(uuid);

// Create the cookie data sealed into content
ContextCookie cookie = createContextCookie(context, itemCode);
```

When using other classification systems, the cookie XML is typically provided as a `org.w3c.dom.Element`. The following code snippet shows the same context details as the sample above, but using a raw XML. The cookie is created using the DOM to create the required XML document. It does not matter how the XML document is created; loaded from a file, created from a string, loaded in from a stream, etc. This example shows the cookie XML being created from a string.

```
String xml =
    "<?xml version=\"1.0\" encoding=\"UTF-8\"?>" +
    "<classifications:ContextCookie
xmlns:classifications=\"http://xmlns.oracle.com/irm/classifications\">" +
    "    <context>" +
    "        <uuid>a4905cd7-7405-469e-b72c-78d11e959b3a</uuid>" +
    "    </context>" +
    "    <itemCode>" +
    "        <value>sample.shtml</value>" +
    "    </itemCode>" +
    "</classifications:ContextCookie>";

java.io.ByteArrayInputStream stream = new
java.io.ByteArrayInputStream(xml.getBytes("utf-8"));

javax.xml.parsers.DocumentBuilderFactory documentBuilderFactory =
javax.xml.parsers.DocumentBuilderFactory.newInstance();

// As the context classification cookie uses namespaces, ensure these are
maintained on parsing the XML
documentBuilderFactory.setNamespaceAware(true);

javax.xml.parsers.DocumentBuilder documentBuilder =
documentBuilderFactory.newDocumentBuilder();

org.w3c.dom.Document document = documentBuilder.parse(stream);
```

3.2.2.2.3 Providing Custom Metadata Custom metadata can be specified during the sealing process. Custom metadata can be provided when creating the `SealingOptions`. Custom metadata is added as an XML element together with a UUID value that can be used to identify the custom data when peeking the sealed content.

```
import static
oracle.irm.engine.content.sealing.CustomDataFactory.createCustomData;
import static
oracle.irm.engine.content.sealing.CustomDataCollectionFactory.createCustomData;
import oracle.irm.engine.content.sealing.CustomData;
...
org.w3c.dom.Element element = ...
UUID uuid = UUID.fromString("7f79d1e8-fc07-464c-8477-834951e07060");

// Create the custom data from the UUID and the XML element
CustomData data = createCustomData(uuid, element);

// Create the set of custom data used when creating the sealing options
java.util.Collection<CustomData> customData = createCustomData(data);
```

Sealed Movie Poster Page

A poster page is the image shown before a sealed movie is started. Oracle IRM Desktop loads the optional poster page from the custom metadata section of the public header. A poster page must be a JPEG or GIF image. The image is provided in the custom data as base 64 encoded data together with the file type.

```
import static
oracle.irm.engine.content.sealing.CustomDataFactory.createCustomData;
import static
oracle.irm.engine.content.sealing.CustomDataCollectionFactory.createCustomData;
import oracle.irm.engine.content.sealing.CustomData;
...
// UUID for poster page image
java.util.UUID posterPageUUID =
java.util.UUID.fromString("6f2c8fba-a2cb-4493-8861-45e5cbda1bac");

// Custom data is base 64 encoded data for image
org.w3c.dom.Element imageElement = document.createElement("item");
imageElement.setTextContent("R0lGODlhhQASAPcAAP.....XfNIQAAAOw==");

CustomData posterPage = createCustomData(posterPageUUID, imageElement);

// UUID for poster page file type - either 'gif' or 'jpeg'
java.util.UUID postPageTypeUUID =
java.util.UUID.fromString("38663feb-5df9-4c14-bd75-b557b6dfea18");

// Custom data is XML document
Element typeElement = document.createElement("item");
type.setTextContent("gif");

CustomData posterPageType = createCustomData(postPageTypeUUID , typeElement);

// Create the set of custom data
java.util.Collection<CustomData> customData = createCustomData(posterPage,
posterPageType);
```

3.3 Peeking

This section contains the following topics:

- Peeking Using the Sealing Server
- Peeking Using the IRM Java API

3.3.1 Peeking Using the Sealing Server

Peeking is the process of extracting metadata from sealed content. This metadata includes the classification details and any custom metadata supplied during the sealing process. Peeking is typically used to extract information from the sealed content without decrypting the file. Peeking is used by Oracle IRM Desktop when sealed file properties are displayed.

The sealing server supports both peeking and validated peek (where the digital signature of the sealed content is validated). In both cases the sealed content is uploaded to the sealing server, the content is examined, and the sealed content metadata is returned to the caller.

3.3.1.1 Uploading Sealed Content

For JAX-WS generated web service proxies, the sealed content is provided as a `DataHandler` parameter. Using a data handler allows the web service stack to stream the binary content to the server without having to load the complete file into memory.

```
javax.activation.DataHandler input = new javax.activation.DataHandler(new
FileDataSource("example.stml"));
```

The data source does not have to be a file.

3.3.1.2 Calling `peek`

A call to the peek method results in the metadata being returned as a `ContentDescription` object. This object contains the classification details, custom metadata and a few other attributes, such as the time the sealed file was created.

```
SealingServices sealingServices = new
SealingServicesService().getSealingServices(new javax.xml.ws.soap.MTOMFeature());

ContentDescription results = sealingServices.peek(input);
```

It is important to enable the MTOM web service feature. This ensures the sealed content is uploaded to the server in the most optimal form. It also avoids `java.lang.OutOfMemoryException` exceptions if the uploaded file is large.

To call the peek operation the authenticated user does not need any rights to access the sealed content.

3.3.1.3 Calling `validatedPeek`

A call to the `validatedPeek` method results in the metadata being returned as a `ContentDescription` object in the same way as peek. If the digital signature has been tampered with, or the file is corrupt, a `ContentParseFault` exception is thrown. This exception will detail the reason for the sealed content parsing failure. A successful invocation of this operation signifies that the metadata signature has been verified.

```
SealingServices sealingServices = new
SealingServicesService().getSealingServices(new javax.xml.ws.soap.MTOMFeature());
```

```
ContentDescription results = sealingServices.validatedPeek(input);
```

To call the validated peek operation, the authenticated user must have the rights to open the sealed content.

3.3.1.4 Examining the Classification

The classification is the most important part of the sealed content metadata. The classification contains the opaque XML document called the classification cookie. The classification cookie is the data used by Oracle IRM Desktop and the Oracle IRM J2EE application when associating rights with content. The classification cookie XML structure is defined by the classification system of the sealed content. The context classification system has an XML structure that includes a UUID to identify the context and a value called the item code which can be used to identify an individual document. The following is a sample context cookie that might appear in sealed content:

```
<?xml version="1.0" ?>
<classifications:ContextCookie
xmlns:classifications="http://xmlns.oracle.com/irm/classifications">
    <context>
        <uuid>588403f9-9cff-4cce-88e4-e030cc57282a</uuid>
    </context>
    <itemCode>
        <value>sample.sdoc</value>
        <time>2007-05-10T12:00:00.000+00:00</time>
    </itemCode>
</classifications:ContextCookie>
```

Rights for the context classification system are expressed using this information, for example:

John can access all documents with a context UUID of `f3cd57c1-f495-48aa-b008-f23afa4d6b07`

or:

Mary can access documents with a context UUID of `f3cd57c1-f495-48aa-b008-f23afa4d6b07` *and an item code value of* `plan001.sdoc` *or* `plan002.sdoc`

The classification metadata also contains the human-readable labels for the classification. There may be multiple labels if the labels have been translated into multiple languages. These labels are used to display a friendly name and description to a user, rather than showing raw computer oriented data from the classification cookie.

3.3.1.5 Reading Labels

The classification contains a set of human-readable strings called labels. The classification labels can be used to inform the user which classification the sealed content was sealed against.

```
Classification classification = results.getClassification();

Label[] labels = classification.getLabels();

if (labels != null) {
    for (Label label : labels) {
        System.out.println(label.getLocale().getDisplayName() + " : " +
```

```
label.getName());
    }
}
```

3.3.1.6 Accessing the Cookie

The classification cookie is defined in the classification XML schema as an `<any>` element. The cookie XML can be accessed from the classification object and is typically returned as a `org.w3c.dom.Element`. The following code snippet shows a context UUID being extracted from a context classification cookie using the DOM.

```
Classification classification = results.getClassification();

org.w3c.dom.Element element = (org.w3c.dom.Element)results.getAny();

org.w3c.dom.NodeList nodes = element.getElementsByTagName("context");
org.w3c.dom.Node node = nodes.item(0);
String uuid = node.getTextContent();
```

3.3.1.7 Large Files

If the file is large there is no need to send the complete file to the sealing server. Peeking only requires the portion of the file that contains the metadata. This portion of the file is dynamic in size, but limited to 1MB in size. A pessimistic view would be to send the first 1MB of the file contents (or the complete contents if this is less than 1MB). In reality the sealed content preamble and metadata are usually a lot smaller, so 16K to 32K is usually sufficient. If the metadata section of the sealed content sent to the sealing server is truncated, the `peek` or `validatedPeek` call will throw a `ContentParseFault`.

3.3.2 Peeking Using the IRM Java API

Peeking is the process of extracting metadata from sealed content. This metadata includes the classification details and any custom metadata supplied during the sealing process. Peeking is typically used to extract information from the sealed content without decrypting the file. Peeking is used by Oracle IRM Desktop when sealed file properties are displayed.

The IRM Java libraries allow peeking to be performed locally. This can be used where performance is an issue and the overhead of sending content to the sealing server is undesirable. The functionality is identical to that provided by remote peeking.

3.3.2.1 Calling `peek`

Local peeking is performed using the `SealingOperations` interface rather than the sealing services web service. Sealed content is provided as an `InputStream` rather than a `DataHandler`.

```
import static oracle.irm.engine.content.sealing.SealingOperationsInstance.peek;

InputStream fileInputStream = new FileInputStream("example.stml");

ContentDescription results = peek(fileInputStream);
```

The result can be examined in the same manner as for remote peeking.

3.4 Resealing

This section contains the following topics:

3.4.1 Resealing Using the Sealing Server

Resealing is the process of altering the custom metadata or editing the encrypted content. Oracle IRM Desktop allows certain formats, such as Microsoft Office, to be edited in sealed form. The process of saving edits is called resealing.

The sealing server supports resealing to update the custom metadata but does not support updating the encrypted content of the sealed file. Content is uploaded to the sealing server, the custom metadata is updated, and the sealed content is returned to the caller.

3.4.1.1 Uploading Content

For JAX-WS generated web service proxies, the content is provided as a `DataHandler` parameter. Using a data handler allows the web service stack to stream the binary content to the server without having to load the complete file into memory.

```
javax.activation.DataHandler input = new javax.activation.DataHandler(new
FileDataSource("example.stml"));
```

The data source does not have to be a file.

3.4.1.2 Calling `reseal`

A call to reseal requires the sealed data (in the form of a `DataHandler`) and the custom data for the update. The following demonstrates how to reseal a sealed file using the `reseal` method adding XML-based custom data to the sealed file.

The XML based custom data is provided as an XML element.

```
Element element = document.createElement("SampleCustomData");
element.setTextContent("Some example custom data provided as an XML element
containing this text");

CustomData data = new CustomData();

// UUID identifies the custom data, this example uses a fixed example UUID value
data.setUuid("7f79d1e8-fc07-464c-8477-834951e07060");

// Custom data is XML document
data.setData(element);
```

Then the `reseal` operation is called to reseal the content and re-sign the metadata.

```
SealingServices sealingServices = new
SealingServicesService().getSealingServices(new javax.xml.ws.soap.MTOMFeature());

DataHandler results = sealingServices.reseal(input, new CustomData[] {data});
```

To call the `reseal` operation, the authenticated user needs rights that allow the reseal feature to be performed for the classification of the sealed content.

3.4.1.3 Extracting the Content

The `DataHandler` class can be used to write out the resealed content to an output stream of the programmer's choice. This example shows the resealed content being written out to a file.

```
java.io.FileOutputStream outputStream = new
java.io.FileOutputStream("example.stml");

results.writeTo(outputStream);

outputStream.close();
```

3.4.2 Resealing Using the IRM Java API

Resealing is the process of altering the custom metadata or editing the encrypted content. Oracle IRM Desktop allows certain formats, such as Microsoft Office, to be edited in sealed form. The IRM Java API allows the custom metadata to be altered. Custom data is provided as an XML element, this code does not show how the XML element is created, just how to use a XML element with custom data. The Java API for XML Code Samples show how to create and manipulate XML documents.

```
org.w3c.dom.Element element = ...
```

Custom data entries are identified with a UUID value.

```
java.util.UUID uuid =
java.util.UUID.fromString("7f79d1e8-fc07-464c-8477-834951e07060");
```

As sealed content can contain multiple custom data entries, each one must have a unique UUID value. This value can be used when peeking the content to identify the custom data. This example uses just one custom data entry.

```
import static
oracle.irm.engine.content.sealing.CustomDataFactory.createCustomData;
import static
oracle.irm.engine.content.sealing.CustomDataCollectionFactory.createCustomData;
import oracle.irm.engine.content.sealing.CustomData;
...
// Create custom data, identified by the UUID value
CustomData data = createCustomData(uuid, element);

// Resealing allows multiple custom data entries, and expects a collection type
java.util.Collection<CustomData> data = createCustomData(data)
```

Once the custom data has been created, the reseal operation can then be called to reseal the content and re-sign the metadata.

```
import oracle.irm.engine.content.sealing.CustomData;
...
// Sealed file provided as a file
java.io.InputStream input = new java.io.FileInputStream("sealed.stml");

// Resealed file written out to a file
java.io.OutputStream output = new java.io.FileInputStream("resealed.stml");

// Reseal the sealed content, altering the custom data
reseal(input, output, data);
```

To call the reseal operation, the authenticated user needs rights that allow the reseal feature to be performed for the classification of the sealed content.

3.5 Reclassifying

This section contains the following topics:

3.5.1 Reclassifying Using the Sealing Server

Reclassifying sealed content is the process of altering the classification of the sealed content. Reclassification usually means re-signing and re-encrypting the content, because most classifications have a dedicated set of cryptography keys. Reclassifying is typically used when content changes sensitivity, for example when a Top Secret document becomes a Company Confidential document.

The sealing server supports reclassifying. Content is uploaded to the sealing server, the classification is updated, and the updated sealed content is returned to the caller.

3.5.1.1 Uploading Content

For JAX-WS generated web service proxies, the content is provided as a `DataHandler` parameter. Using a data handler allows the web service stack to stream the binary content to the server without having to load the complete file into memory.

```
javax.activation.DataHandler input = new javax.activation.DataHandler(new
FileDataSource("example.stml"));
```

The data source does not have to be a file.

3.5.1.2 Calling `reclassify`

A call to reclassify requires the sealed data (in the form of a `DataHandler`) and the new classification details. Refer to the sealing example for details about how to specify a classification in code.

```
SealingServices sealingServices = new
SealingServicesService().getSealingServices(new javax.xml.ws.soap.MTOMFeature());

DataHandler results = sealingServices.reclassify(input,classification);
```

It is important to enable the MTOM web service feature. This ensures the sealed content is uploaded to the server in the most optimal form. It also avoids `java.lang.OutOfMemoryException` exceptions if the uploaded file is large.

To call the reclassify operation, the authenticated user needs either:

- Rights that allow the *copy to* feature for the source classification with a trusted destination that allows the target classification, and rights that allow the *seal* feature for the target classification.

  In the context classification system, this means the user has to have a role that has export constraints set as *trusted* with the target context being a trusted context of the source context, or that the role has export constraints set as *none*.

- Rights that allow the *unseal* feature for the specified classification and the *seal* feature for the target classification.

  In the context classification system, this means the user has to have a role that has export constraints set as *none*.

When using the Oracle IRM Server Management Console, the *copy to* and *unseal* features are enabled and controlled using the export constraints defined on a role.

3.5.1.3 Extracting the Content

The `DataHandler` class can be used to write out the resealed content to an output stream of the programmer's choice. This example shows the resealed content being written out to a file.

```
java.io.FileOutputStream outputStream = new
java.io.FileOutputStream("example.stml");

results.writeTo(outputStream);

outputStream.close();
```

3.5.2 Reclassifying Using the IRM Java API

Reclassifying sealed content is the process of altering the classification of the sealed content. Reclassification usually means re-signing and re-encrypting the content, because most classifications have a dedicated set of cryptography keys. Reclassifying is typically used when content changes sensitivity, for example when a Top Secret document becomes a Company Cconfidential document. A call to reclassify requires the sealed data in the form of a `InputStream`, and the new classification details. The resultant reclassified sealed content is written to the provided `OutputStream`.

```
// Sealed content is provided as a file
java.io.InputStream input = new java.io.FileInputStream("sealed.stml");

// Reclassified content is written to a file
java.io.InputStream output = java.io.new FileInputStream("reclassified.stml");

// Reclassify the sealed content to the provided classification
reclassify(input, output, classification);
```

Refer to the sealing example for details about how to create a classification in code.

To call the reclassify operation, the authenticated user needs either:

- Rights that allow the Copy To feature for the source classification, with a trusted destination that allows the target classification, and rights that allow the seal feature for the target classification.
- Rights that allow the Unseal feature for the specified classification and the seal feature for the target classification.

> **Unseal and Copy To Features:** When using the Oracle IRM Server Management Console, the Copy To and Unseal features are enabled and controlled using the export constraints defined on a role.

3.6 Unsealing

This section contains the following topics:

3.6.1 Unsealing Using the Sealing Sever

Unsealing is the process of converting sealed content back into the original, plaintext content. Unsealing is typically used to convert sealed content that is no longer

sensitive back into normal content. Unsealing is an operation that is supported by both Oracle IRM Desktop and the sealing server.

The sealing server supports unsealing. The sealed content is uploaded to the sealing server, the content is decrypted, and the unsealed content is returned to the caller.

3.6.1.1 Uploading Sealed Content

For JAX-WS generated web service proxies, the sealed content is provided as a `DataHandler` parameter. Using a data handler allows the web service stack to stream the binary content to the server without having to load the complete file into memory.

```
javax.activation.DataHandler input = new javax.activation.DataHandler(new
FileDataSource("example.stml"));
```

The data source does not have to be a file.

3.6.1.2 Calling `unseal`

A call to the unseal method results in the unsealed data being returned as a `javax.activation.DataHandler`. This object can be used to stream the unsealed data into a file or buffer.

```
SealingServices sealingServices = new
SealingServicesService().getSealingServices(new javax.xml.ws.soap.MTOMFeature());

javax.activation.DataHandler results = sealingServices.unseal(input);
```

It is important to enable the MTOM web service feature. This ensures the sealed content is uploaded to the server in the most optimal form. It also avoids `java.lang.OutOfMemoryException` exceptions if the uploaded file is large.

To call the unseal operation, the authenticated user needs rights that allow the unseal feature to be performed for the classification of the sealed content.

When using the Oracle IRM Server Management Console, the unseal feature is enabled when a role has export constraints of *none*.

3.6.1.3 Extracting the Content

The `DataHandler` class can be used to write out the unsealed content to an output stream of the programmer's choice. This example shows the unsealed content being written out to a file.

```
java.io.FileOutputStream outputStream = new
java.io.FileOutputStream("example.html");

results.writeTo(outputStream);

outputStream.close();
```

3.6.2 Unsealing Using the IRM Java API

Unsealing is the process of converting sealed content back into the original, plaintext content. Unsealing is typically used to convert sealed content that is no longer sensitive back into normal content. Unsealing is an operation that is supported by both Oracle IRM Desktop and the sealing server. The IRM Java API unseal operation takes an input stream to the sealed data and writes the plain text results to an output stream. The sealed content metadata is also returned from this operation. In the following example, a `FileInputStream` and `FileOutputStream` are used to read and write the sealed and unsealed data.

```
import oracle.irm.engine.content.sealing.SealingOperationsInstance.unseal;
import oracle.irm.engine.content.sealing.ContentDescription;
...
// Sealed content provided as an file
java.io.InputStream input = new java.io.FileInputStream("example.sdoc");

// Unsealed content written out to a file
java.io.OutputStream output = new java.io.FileOutputStream("example.doc");

// Unseal the content and get the sealed meta-data as a result.
ContentDescription results = unseal(input, output);
```

The results of the unseal operation can be used to determine the classification of the sealed content.

```
import oracle.irm.engine.core.classification.Classification;
...
// Extract the classification details
Classification classification = results.getClassification();
```

Or to examine the custom data embedded in the sealed content. .

```
import oracle.irm.engine.content.sealing.CustomData;
...
// Extract the custom data details
java.util.Collection<CustomData> customData = results.getCustomData();
```

To call the unseal operation, the authenticated user needs rights that allow the unseal feature to be performed for the classification of the sealed content.

Unseal Feature: When using the Oracle IRM Server Management Console, the unseal feature is enabled when a role has export constraints of `none`.

4

Working with Domains, Contexts, Roles, and Rights

This section contains the following topics:

- Concepts
- Examples

4.1 Concepts

This section contains the following topics:

- Domains
- Context Templates
- Contexts
- Roles (Document Roles)
- Rights (Document Rights)

Access to context-classified documents is governed by rights, such as the right to open a document, the right to print it, and the right to copy information from it and paste it into another document. The rights are defined and assigned centrally by administrators, who group combinations of rights and end user identities into one or more "contexts".

4.1.1 Domains

A domain is the top level entity that contains document roles and context templates.

4.1.1.1 `DomainRef`

Domains are identified by a UUID value. Web services operations that need to identify a domain use the `DomainRef` type. A `DomainRef` contains only the information required to uniquely identify a domain; the UUID. The other properties of a domain, such as its labels, are not part of a `DomainRef` type.

```
DomainRef domainReference = new DomainRef();
domainReference.setUuid("29499ec3-4ded-4138-9ea4-cc1f5dbf8db1");
```

An example method that uses a `DomainRef` type is the `delete domain` method:

```
DomainOperations domainOperations = new
DomainOperationService().getDomainOperations();
domainOperations.deleteDomain(domainReference);
```

4.1.1.2 Domain

A `Domain` contains all the information about a domain, including the UUID. For operations that require or return all the domain properties, a `Domain` is used.

An example that uses the `Domain` type is the `list domains` method:

```
DomainOperations domainOperations = new
DomainOperationService().getDomainOperations();
Domain[] results = domainOperations.listDomains();
```

4.1.2 Context Templates

Context templates are identified by a UUID value within the owning domain. Web services operations that need to identify a template use the `ContextTemplateRef` type. A `ContextTemplateRef` contains the information required to uniquely identify a template: the UUID and the owning domain. The other properties of a template, such as its labels and roles, are not part of a `ContextTemplateRef` type.

4.1.2.1 ContextTemplateRef

```
ContextTemplateRef templateReference = new ContextTemplateRef();
templateReference.setUuid("96c512a7-e44b-47d5-a70d-0ef2c0283f25");
templateReference.setDomain(domainReference);
```

It is valid for two templates in different domains to have the same UUID value. The templates that are automatically installed when the first domain is created all have fixed UUID values. For example, the standard template has the UUID value `474dbb07-718b-4c4e-8f43-d2b723469573`. Using these predefined UUID values means there is no need to look up a context template UUID before creating a ContextTemplateRef type.

4.1.2.2 ContextTemplate

A `ContextTemplate` contains all the information about a context template, including the UUID and domain. For operations that require or return all the context template properties, a `ContextTemplate` is used.

4.1.3 Contexts

Contexts are identified by a UUID value. Web services operations that need to identify a context use the `ContextInstanceRef` type. A `ContextInstanceRef` contains the information required to uniquely identify a context, the UUID. The other properties of a context, such as labels and item exclusions, are not part of a `ContextInstanceRef` type.

4.1.3.1 ContextInstanceRef

```
ContextInstanceRef contextReference = new ContextInstanceRef();
contextReference.setUuid("a3c41a86-a1b4-4d5d-9fe5-f4077bbc2b17");
```

4.1.3.2 ContextInstance

A `ContextInstance` contains all the information about a context, including the UUID and labels. For operations that require or return all the context properties, a `ContextInstance` is used.

4.1.4 Roles (Document Roles)

A document role defines a set of criteria that specify how sealed content can be used. A document role is assigned to a user or group, allowing the user to use sealed content in the way the role defines.

4.1.4.1 DocumentRoleRef

Document roles are identified by a UUID value within the owning domain. Web services operations that need to identify a role use the `DocumentRoleRef` type. A `DocumentRoleRef` contains the information required to uniquely identify a role; the UUID and the owning domain. The other properties of a role, such as its labels and features, are not part of a `DocumentRoleRef` type.

```
DocumentRoleRef roleReference = new DocumentRoleRef();
roleReference.setUuid("96c512a7-e44b-47d5-a70d-0ef2c0283f25");
roleReference.setDomain(domainReference);
```

It is valid for two roles in different domains to have the same UUID value. The roles that are automatically installed when the first domain is created all have fixed UUID values. For example, the contributor role has the UUID value `a456140d-24dc-4cc2-8f23-1a72fb6c2d81`. Using these predefined UUID values means there is no need to look up a context template UUID before creating a ContextTemplateRef type.

4.1.4.2 DocumentRole

A `DocumentRole` contains all the information about a role, including the UUID and domain. For operations that require or return all the document role properties a `DocumentRole` is used.

4.1.5 Rights (Document Rights)

Rights are identified by a UUID value. Web services operations that need to identify a right use the `DocumentRightRef` type. A `DocumentRightRef` contains the information required to uniquely identify a right, the UUID. The other properties of a right, such as assigned account and role, are not part of a `DocumentRightRef` type.

4.1.5.1 DocumentRightRef

```
DocumentRightRef rightReference = new DocumentRightRef();
rightReference.setUuid("35aa8611-abff-43e3-a5ae-ffe71345d9d4");
```

4.1.5.2 DocumentRight

A `DocumentRight` contains all the information about a right, including the UUID, account and role. For operations that require or return all the right properties, a `DocumentRight` is used.

4.2 Examples

This section contains the following topics:

- Creating a Context from a Template
- Searching for Journal Entries
- Assigning a Role
- Listing the Rights Assigned to a User or Group

- Unassigning a Role

The context operations web service provides operations that a domain manager, inspector or context manager would typically perform. This includes creating contexts, altering context labels, and adding or removing context managers.

The document right operations web service provides operations that allow a context manager to manage the rights users have within a context. Document right operations include assigning rights, checking in rights, and listing rights.

A document role can be assigned, within a context, to one or more accounts. This can be performed by users that have the Context Manager role within the context. An account can only have one role assigned within a context.

4.2.1 Creating a Context from a Template

A context is created from a context template. The template defines the structure of the context and what roles are available to assign to users and groups. Only active templates can be used when creating contexts. Changes to the template after the context is created are dynamically picked up in the context. For example, adding a role to the template makes the role available to the context.

4.2.1.1 Calling `createContextFromTemplate`

When creating a context, the relevant context template must be identified using a ContextTemplateRef type. This type includes the template UUID and owning domain. The domain created on installation has a fixed UUID value of `dcfef562-971d-401b-81f9-86700573bf8b`. If other domains are used, the domain UUID can be found by using operations such as `listDomains`.

```
DomainRef domainReference = new DomainRef();
domainReference.setUuid("dcfef562-971d-401b-81f9-86700573bf8b");
```

The standard templates installed in the installation domain also have fixed UUID values.

- `474dbb07-718b-4c4e-8f43-d2b723469573` for the standard context template.
- `1a05b98d-b415-4c38-9b7a-9d0ad19ee0e9` for the export context template.

If other templates are used, the template UUID can be obtained using the `listActiveTemplates` operations.

```
ContextTemplateRef templateRef = new ContextTemplateRef();
templateRef.setUuid("474dbb07-718b-4c4e-8f43-d2b723469573");
templateRef.setDomain(domainReference);
```

Once the template has been identified, a context can be created. The authenticated user will automatically be made the context manager.

```
ContextOperations contextOperations = new
ContextOperationsService().getContextOperations();

Label english = new Label();
english.setLocale("en");
english.setName("Top Secret");

Label german = new Label();
german.setLocale("de");
german.setName("Strenges Geheimnis");
```

```
ContextInstance context = contextOperations.createContextFromTemplate(
    null, // automatically generate a UUID value for the context
    templateRef,
    new Label[] { english, german },
    Visibility.DOMAIN,
    null); // no additional context managers
```

4.2.2 Searching for Journal Entries

The context journal contains records of actions performed on sealed content of the context classification system. This information is available to administrators in the Reports tab of the Oracle IRM Server Management Console. The context journal can be searched for activity on content for the specified accounts and/or document items. This search is restricted to the contexts available to the caller. That is, the caller must be a context manager or inspector.

4.2.2.1 Calling `searchJournal`

Searching for journal entries may produce a large result set. For this reason a page range (starting from 1) must be provided.

```
PageRange pageRange = new PageRange();
pageRange.setFirst(1);
pageRange.setLast(100);
```

A time range to filter the search is also required. The following example is a time range for the last twenty-four hours.

```
Date end = new Date();

// Use a calendar to work out the time range
Calendar calendar = Calendar.getInstance();

calendar.setTime(end);
calendar.add(Calendar.DAY_OF_MONTH, -1);

Date begin = calendar.getTime();

TimeRange timeRange = new TimeRange().
timeRange.setBegin(begin);
timeRange.setEnd(end);
```

If no sort order is specified, the results are sorted by time.

```
ContextOperations contextOperations = new
ContextOperationsService().getContextOperations();

ContextJournalEntry[] journalResults = contextOperations.searchJournal(
    null, // no accounts filter
    null, // no item codes filter
    timeRange,
    pageRange,
    null); // no sorting details
```

4.2.3 Assigning a Role

A document role can be assigned to a user or group. Document roles are identified with a UUID and a domain. The domain created on installation has a fixed UUID value of `dcfef562-971d-401b-81f9-86700573bf8b`. If other domains are used, the domain UUID can be found by using operations such as `listDomains`.

```
DomainRef domainReference = new DomainRef();
domainReference.setUuid("dcfef562-971d-401b-81f9-86700573bf8b");
```

The standard roles installed in the installation domain also have fixed UUID values.

- `a456140d-24dc-4cc2-8f23-1a72fb6c2d81` for the contributor role.
- `6dbed6c1-6a45-4da1-9aff-c8f1b4d856c4` for the contributor with export role.
- `b68278a1-70d1-4f24-aae2-2803729a6674` for the item reader role.
- `48c2e03c-9cd3-4bb1-91db-3eaa4564adc2` for the reader role.
- `37646b77-aee3-418c-8664-4101fa7b44df` for the reader with export role.
- `e70daaa1-0c27-4f8e-aa8b-d8dfc34c4579` for the reader no print role.
- `7d25eda0-2641-445f-9c94-45798165b262` for the reviewer role.

If other roles are used, the role UUID can be obtained using the `listDocumentRoles` operations.

```
DocumentRoleRef roleReference = new DocumentRoleRef();
roleReference.setUuid("a456140d-24dc-4cc2-8f23-1a72fb6c2d81");
roleReference.setDomain(domainReference);
```

The context is also identified with a UUID. If the context was created using the web services, the UUID value may already be known. If not, use the `listContexts` operations to identity the context required.

```
ContextInstanceRef contextReference = new ContextInstanceRef();
contextReference.setUuid("9c8d7f1f-9819-4c8e-833f-380f3141e2b6");
```

The user or group can be identified by GUID or name. See "Working with Users and Groups" on page 5-1 for more details.

```
AccountRef user1 = new AccountRef();
user1.setUuid("9c8d7f1f-9819-4c8e-833f-380f3141e2b6");

AccountRef user2 = new AccountRef();
user2.setUuid("urn:user:john.smith");
```

Once the context, role, and accounts have been identified, the role can be assigned.

```
DocumentRightOperations rightOperations = new
DocumentRightOperationsService().getDocumentRightOperations();

rightOperations.assignRole(
    contextInstanceRef,
    roleRef,
    new AccountRef[] {user1, user2},
    null); // no item constraints
```

If the role is item locked, items can also be specified. The item code values may be well known or can be obtained by peeking sealed content. See "Peeking" on page 2-3.

```
ItemCode itemCode = new ItemCode();
itemCode.setValue("example.stml");

rightOperations.assignRole(
    contextInstanceRef,
    roleRef,
    new AccountRef[] {user1, user2},
```

```
    new ItemCode[] {itemCode});
```

4.2.4 Listing the Rights Assigned to a User or Group

A context manager or inspector can list the rights for the contexts that they are allowed to see. Rights listed for a user or group include rights obtained indirectly through group membership.

```
AccountRef user = new AccountRef();
user.setUuid("urn:user:john.smith");

DocumentRightOperations rightOperations = new
DocumentRightOperationService().getDocumentRightOperations();

// Get all of the rights assigned to the account
DocumentRight[] rights = rightOperations.listRightsByAccount(user);
```

Or for a group:

```
AccountRef group= new AccountRef();
group.setUuid("urn:group:everyone");

DocumentRightOperations rightOperations = new
DocumentRightOperationService().getDocumentRightOperations();

// Get all of the rights assigned to the account
DocumentRight[] rights = rightOperations.listRightsByAccount(group);
```

4.2.5 Unassigning a Role

A context manager can remove roles that have already been assigned within the context to an account. This is performed using the `unassignRights` method. The assignment of a document role to an account is stored as a document right identified by a UUID.

```
DocumentRightRef rightRef = new DocumentRightRef();
rightRef.setUuid("ff34e6f9-364b-550d-dfa7-bdf56b0c8188");

DocumentRightOperations rightOperations = new
DocumentRightOperationsService().getDocumentRightOperations();

rightOperations.unassignRights(new DocumentRightRef[] { rightRef });
```

The UUID for the relevant right can be obtained by using methods such as `listRightsByAccount` or `listRightsByContext`.

5 Working with Users and Groups

This section contains the following topics:

5.1 The `AccountRef` Type

This section contains the following topics:

5.1.1 About the `AccountRef` Type

The `AccountRef` type contains all the information needed to identify a user or group. Typically this type contains a GUID value, but also allows the user or group name to be used. The following code snippets show how to create an `AccountRef` using a GUID or a user name or a group name.

5.1.2 Creating an `AccountRef` Using a GUID

```
AccountRef accountReference = new AccountReference();
accountReference.setUuid("c9d55dc2-92f6-405d-aa52-ff12ab2792ef");
```

The GUID format may differ depending on the identity store used.

5.1.3 Creating an `AccountRef` Using a User Name

The user name must be URL encoded. For example, 'John Smith' could be encoded as 'John+Smith'.

```
AccountRef accountReference = new AccountReference();
accountReference.setUuid("urn:user:john.smith");
```

5.1.4 Creating an `AccountRef` Using a Group Name

The group name must be URL encoded.

```
AccountRef accountReference = new AccountReference();
accountReference.setUuid("urn:group:everyone");
```

5.2 Obtaining User and Group Names

The IRM web services typically returns user and group GUID values rather than user and group names. Rather than having to perform direct lookups of user and group names against the identity store, the Oracle IRM web services includes a user and group lookup mechanism. The web service method takes a list of GUID values (in the form of `AccountRef` types) and returns the user or group name. The following code example shows the domain administrator user names being looked up with the `listAccountDetails` web service operation.

```
DomainOperations domainOperations = new
DomainOperationsService().getDomainOperations();

AccountRef[] accounts =
domainOperations.listDomainAdministrators(domainReference);

Account[] results = domainOperations.listAccountDetails(accounts);

for(Account account : results)
{
    System.out.println(account.getName() + " : " + account.getType());
}
```

To call this method the authenticated user must have the domain administrator, domain manager, inspector, or context manager roles in any domain.

6

Code Samples for Web Services

This section contains the following topics:

6.1 Web Services

The following table lists all the web services available from a deployed Oracle IRM J2EE application.

Name	Description	WSDL Document URL / Web Service Endpoint
Sealing Services	Provides sealed content processing operations such as sealing, unsealing and reclassification.	http://irm.example.com/irm_sealing/sealing_services?wsdl https://irm.example.com/irm_sealing/sealing_services
Desktop Services	Provides license management operations for the sealing server.	http://irm.example.com/irm_sealing/desktop_services?wsdl https://irm.example.com/irm_sealing/desktop_services
Context Operations	Provides operations for creating, editing and deleting contexts, context managers and inspectors.	http://irm.example.com/irm_services/context_operations?wsdl https://irm.example.com/irm_services/context_operations
Context Template Operations	Provides operations for creating, editing and deleting context templates.	http://irm.example.com/irm_services/context_template_operations?wsdl https://irm.example.com/irm_services/context_template_operations
Document Right Operations	Provides operations for assigning roles, unassigning rights and altering item restrictions.	http://irm.example.com/irm_services/document_right_operations?wsdl https://irm.example.com/irm_services/document_right_operations

Name	Description	WSDL Document URL / Web Service Endpoint
Document Role Operations	Provides operations for creating, editing and deleting document roles.	https://irm.example.com/irm_services/document_role_operations?wsdl http://irm.example.com/irm_services/document_role_operations
Domain Operatons	Operations for creating, altering and deleting domains.	https://irm.example.com/irm_services/domain_operations?wsdl http://irm.example.com/irm_services/domain_operations

6.2 Using JDeveloper Generated Web Services Proxies

This section contains the following topics:

6.2.1 Introduction

The following section provides sample code that can be used with JAX-WS web service proxies generated using JDeveloper 11g.

6.2.2 Using the Samples

The JDeveloper 11g sample code comes packaged with a pre-generated set of web service proxy code so there is no need to generate the web service proxy code. The code samples in this document do not show this generated code, but assume this code is present in a package called `generated`. The easiest way to use the samples is to import them directly into a new or existing JDeveloper 11g project using the File > Import > Java Source menu.

> **Note:** Before running each sample, check the code to see what command line arguments the sample requires.

6.2.3 Generating a Web Service Proxy

If the web service proxy code needs to be generated by hand, these are the steps to follow within JDeveloper 11g:

1. From the File menu, select New, then Business Tier, then Web Services, then Web Service Proxy.
2. Select a JAX-WS Style client and press Next.
3. Enter the required WSDL document URL as listed in the Web Services section of this document.

   For example, http://irm.example.com/irm_sealing/sealing_services?wsdl

   There is no need to copy the WSDL into the project.
4. Press Next.
5. Select Run against a service deployed to an external server.
6. Press Finish.

6.2.4 Creating a Domain

The following code demonstrates how to create a domain. The sample code uses a fixed domain UUID so that all sample code can work against a known domain. A new domain would typically be given a new random UUID value. The authenticated user becomes the domain administrator. When a domain is created, a set of human-readable labels can be given to the domain for the target language audience.

Example 6–1

```
import generated.Domain;
import generated.DomainOperations;
import generated.DomainOperationsService;
import generated.Label;

import java.net.Authenticator;
import java.net.PasswordAuthentication;
import java.util.Locale;
import java.util.Map;

import javax.xml.ws.BindingProvider;

public class SaveNewDomain {

    public static void main(String[] args) throws Exception {
```

```
        final String endpointAddress = args[0];
        final String username = args[1];
        final String password = args[2];

        // Configure an authenticator to provide the credentials
        // for the web service
        Authenticator.setDefault(new Authenticator() {
            @Override
            protected PasswordAuthentication getPasswordAuthentication() {
                return new PasswordAuthentication(username,
password.toCharArray());
            }
        });
```

6.2.5 Creating a Role

The following code demonstrates how to create a role. The sample code uses a fixed role UUID so that all sample code can work with a known role. A new role would typically be given a new random UUID value. The sample role is set up to allow all the content operations required by the sample code. When assigned to a user, this role allows sealing, unsealing, resealing and (validated) peeking. This is done by a providing an appropriate set of features and export constraints.

Example 6–2

```
import generated.DocumentRole;
import generated.DocumentRoleExportConstraints;
import generated.DocumentRoleOperations;
import generated.DocumentRoleOperationsService;
import generated.DomainRef;
import generated.Feature;
import generated.FeatureUse;
import generated.ItemConstraintsType;
import generated.Label;
import generated.LicenseCriteriaStorage;
import generated.TimePeriod;
import generated.TimePeriodUnits;

import java.net.Authenticator;
import java.net.PasswordAuthentication;
import java.util.List;
import java.util.Locale;
import java.util.Map;

import javax.xml.ws.BindingProvider;

public class SaveNewRole {

    public static void main(String[] args) throws Exception {

        final String endpointAddress = args[0];
        final String username = args[1];
        final String password = args[2];

        // Configure an authenticator to provide the credentials
        // for the web service
        Authenticator.setDefault(new Authenticator() {
            @Override
            protected PasswordAuthentication getPasswordAuthentication() {
```

```
                return new PasswordAuthentication(username,
password.toCharArray());
            }
        });

        // Get the document role operations web service
        DocumentRoleOperationsService service = new
DocumentRoleOperationsService();

        DocumentRoleOperations roleOperations =
service.getDocumentRoleOperations();

        // Set the end point address
        Map<String, Object> requestContext =
((BindingProvider)roleOperations).getRequestContext();

        requestContext.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
endpointAddress);

        // The Role UUID value - that identifies this role within the domain -
        // is fixed for sample code
        DocumentRole role = new DocumentRole();

        role.setUuid("ee82c3f9-152b-440d-afd7-dbf36b0c8188");

        // Role has one English label
        Label label = new Label();

        label.setLocale(Locale.ENGLISH.toString());
        label.setName("Sample Role");
        label.setDescription("This is a role created from sample code.");

        role.getLabels().add(label);

        // This role allows the user to access content while offline by persisting
licenses on the desktop
        role.setStorage(LicenseCriteriaStorage.PERSISTENT);

        // This role allows content to be saved in the clear (unsealing and
copying)
        role.setExportConstraints(DocumentRoleExportConstraints.NONE);

        // This role allows opening,
        Feature open = new Feature();

        open.setId("oracle.irm.generic.Open");
        open.setUse(FeatureUse.IMMEDIATE);
        open.setRecord(false);

        // sealing,
        Feature seal = new Feature();

        seal.setId("oracle.irm.generic.Seal");
        seal.setUse(FeatureUse.IMMEDIATE);
        seal.setRecord(false);

        // and resealing.
        Feature reseal = new Feature();

        reseal.setId("oracle.irm.generic.Reseal");
```

```
        reseal.setUse(FeatureUse.IMMEDIATE);
        reseal.setRecord(false);

        List<Feature> features = role.getFeatures();

        features.add(open);
        features.add(seal);
        features.add(reseal);

        // Role allows document exclusions to be listed, by default all items are
allowed
        role.setItemConstraints(ItemConstraintsType.EXCLUSIONS);

        // This role allows content to be opened for one hour before refreshing
the rights from the server
        // This role has no additional time constraints
        TimePeriod value = new TimePeriod();

        value.setAmount(1);
        value.setUnits(TimePeriodUnits.HOURS);

        role.setRefreshPeriod(value);

        // Domain UUID is fixed for sample code
        DomainRef domain = new DomainRef();

        domain.setUuid("6fab93fd-2858-461a-a0b3-34e261dbf8fd");

        // Save the new role
        roleOperations.saveNewRole(domain, role);
    }
}
```

6.2.6 Creating a Context Template

The following code demonstrates how to create a context template. The sample code uses a fixed template UUID so that all sample code can work with a known template. A new template would typically be given a new random UUID value. The sample template has one role and is active. This template is used to create contexts in the create context code sample.

Example 6–3

```
import static
oracle.irm.j2ee.jws.rights.context.ContextTemplateOperations.getContextTemplateOpe
rationsEndpoint;
```

6.2.7 Creating a Context

The following code demonstrates how to create a context from a context template. The sample code uses a fixed context template reference (information that identifies the template) and provides a fixed UUID value for the new context. The authenticated user becomes the context manager. The context is created with two labels, English and German. This context is used in the sample code that assigns a role, as well as the sealing, unsealing, resealing, reclassification and peeking code samples.

Example 6–4

```
import generated.ContextInstanceVisibility;
```

```
import generated.ContextOperations;
import generated.ContextOperationsService;
import generated.ContextTemplateRef;
import generated.DomainRef;
import generated.Label;

import java.net.Authenticator;
import java.net.PasswordAuthentication;
import java.util.ArrayList;
import java.util.List;
import java.util.Locale;
import java.util.Map;

import javax.xml.ws.BindingProvider;

public class CreateContextFromTemplate {

    public static void main(String[] args) throws Exception {

        final String endpointAddress = args[0];
        final String username = args[1];
        final String password = args[2];

        // Configure an authenticator to provide the credentials
        // for the web service
        Authenticator.setDefault(new Authenticator() {
            @Override
            protected PasswordAuthentication getPasswordAuthentication() {
                return new PasswordAuthentication(username,
password.toCharArray());
            }
        });

        // Get the content operations endpoint
        ContextOperationsService service = new ContextOperationsService();

        ContextOperations contextOperations = service.getContextOperations();

        // Set the end point address
        Map<String, Object> requestContext =
((BindingProvider)contextOperations).getRequestContext();

        requestContext.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
endpointAddress);

        // Domain UUID is fixed for sample code
        DomainRef domain = new DomainRef();

        domain.setUuid("6fab93fd-2858-461a-a0b3-34e261dbf8fd");

        // Create the context template reference
        ContextTemplateRef templateRef = new ContextTemplateRef();

        templateRef.setDomain(domain);

        // Context Template UUID is for the "standard" template automatically
installed with a domain
        templateRef.setUuid("930876e6-a505-4a10-8d93-bc43d9a37c23");

        templateRef.setDomain(domain);
```

```
        // Context UUID is fixed for sample code
        String contextUUID = "46f910d9-dd30-476e-b060-4d01f88f8b05";

        // Context has two labels, English and German
        Label english = new Label();

        english.setLocale(Locale.ENGLISH.toString());
        english.setName("Sample Classification");
        english.setDescription("Created from sample code.");

        Label german = new Label();

        german.setLocale(Locale.GERMAN.toString());
        german.setName("Beispielklassifikation");
        german.setDescription("Verursacht vom Beispielcode.");

        List<Label> labels = new ArrayList<Label>();

        labels.add(english);
        labels.add(german);

        // Create a context based on that template
        contextOperations.createContextFromTemplate(
            contextUUID, // context UUID value
            templateRef, // context template
            labels, // labels
            ContextInstanceVisibility.DOMAIN, // visibility
            null); // additional context managers
    }
}
```

6.2.8 Assigning a Role to a User

The following code demonstrates how to assign a role to a user. To assign a role, the role, context and user or group must be specified. If the role is restricted to individual items then items can also be specified as in the assign role method.

Example 6–5

```
import generated.AccountRef;
import generated.ContextInstanceRef;
import generated.DocumentRightOperations;
import generated.DocumentRightOperationsService;
import generated.DocumentRoleRef;
import generated.DomainRef;

import java.net.Authenticator;
import java.net.PasswordAuthentication;
import java.net.URLEncoder;
import java.util.Collections;
import java.util.Map;

import javax.xml.ws.BindingProvider;

public class AssignRole {

    public static void main(String[] args) throws Exception {

        final String endpointAddress = args[0];
```

```
        final String username = args[1];
        final String password = args[2];

        // Configure an authenticator to provide the credentials
        // for the web service
        Authenticator.setDefault(new Authenticator() {
            @Override
            protected PasswordAuthentication getPasswordAuthentication() {
                return new PasswordAuthentication(username,
password.toCharArray());
            }
        });

        // Get the document right operations endpoint
        DocumentRightOperationsService service = new
DocumentRightOperationsService();

        DocumentRightOperations rightOperations =
service.getDocumentRightOperations();

        // Set the end point address
        Map<String, Object> requestContext =
((BindingProvider)rightOperations).getRequestContext();

        requestContext.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
endpointAddress);

        // Domain UUID is fixed for sample code
        DomainRef domainRef = new DomainRef();

        domainRef.setUuid("6fab93fd-2858-461a-a0b3-34e261dbf8fd");

        // Document Role UUID is for the "Sample Role" role
        DocumentRoleRef roleRef = new DocumentRoleRef();

        roleRef.setUuid("ee82c3f9-152b-440d-afd7-dbf36b0c8188");
        roleRef.setDomain(domainRef);

        // Context UUID is fixed for sample code
        ContextInstanceRef contextInstanceRef = new ContextInstanceRef();

        contextInstanceRef.setUuid("46f910d9-dd30-476e-b060-4d01f88f8b05");

        // Reference the account by user name
        AccountRef accountRef = new AccountRef();

        accountRef.setUuid("urn:user:" + URLEncoder.encode(username, "utf-8"));

        // Assign the role to the account
        rightOperations.assignRole(
            contextInstanceRef,
            roleRef,
            Collections.singletonList(accountRef),
            null); // no item constraints
    }
}
```

6.2.9 Listing Rights Assigned to a User or Group

The following code demonstrates how to list the rights that have been assigned to a user or group. The code displays the role label and the context UUID from each right.

Example 6–6

```
import generated.AccountRef;
import generated.DocumentRight;
import generated.DocumentRightOperations;
import generated.DocumentRightOperationsService;
import generated.ItemCode;

import java.net.Authenticator;
import java.net.PasswordAuthentication;
import java.net.URLEncoder;
import java.util.Collection;
import java.util.Map;

import javax.xml.ws.BindingProvider;

public class ListRightsByAccount {

    public static void main(String[] args) throws Exception {

        final String endpointAddress = args[0];
        final String username = args[1];
        final String password = args[2];

        // Configure an authenticator to provide the credentials
        // for the web service
        Authenticator.setDefault(new Authenticator() {
            @Override
            protected PasswordAuthentication getPasswordAuthentication() {
                return new PasswordAuthentication(username,
password.toCharArray());
            }
        });

        // Get the document right operations endpoint
        DocumentRightOperationsService service = new
DocumentRightOperationsService();

        DocumentRightOperations rightOperations =
service.getDocumentRightOperations();

        // Set the end point address
        Map<String, Object> requestContext =
((BindingProvider)rightOperations).getRequestContext();

        requestContext.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
endpointAddress);

        // Reference the account by user name, allowed formats are
        //  urn:user:xxxx
        //  urn:group:xxxx
        //  00000000-0000-0000-0000-000000000000
        AccountRef accountRef = new AccountRef();

        accountRef.setUuid("urn:user:" + URLEncoder.encode(username, "utf-8"));
```

```
        // Get all of the rights assigned to the account
        Collection<DocumentRight> rights =
rightOperations.listRightsByAccount(accountRef);

        // Display a summary of each right
        for (DocumentRight right : rights) {
            System.out.println("Account: " + right.getAccount().getUuid());
            System.out.println("Context: " + right.getContext().getUuid());
            System.out.println("Role: " + right.getRole().getUuid());

            // Show items
            Collection<ItemCode> itemCodes = right.getItemCodes();

            if (itemCodes != null) {
                for (ItemCode itemCode : itemCodes) {
                    System.out.println(" ItemCode: " + itemCode.getValue());
                }
            }
        }
    }
}
```

6.2.10 Altering the Role Assigned to a User or Group

The following code demonstrates how to alter a role assignment using the reassignRole method over web services. The sample code adds an item code exclusion to a role assignment. Typically this method is used to alter the role, but as the sample code only has one demonstration role it shows how to alter the item restrictions.

Example 6–7

```
import generated.AccountRef;
import generated.DocumentRight;
import generated.DocumentRightOperations;
import generated.DocumentRightOperationsService;
import generated.DocumentRightRef;
import generated.DocumentRoleRef;
import generated.DomainRef;
import generated.ItemCode;

import java.net.Authenticator;
import java.net.PasswordAuthentication;
import java.net.URLEncoder;
import java.util.Collections;
import java.util.List;
import java.util.Map;

import javax.xml.ws.BindingProvider;

public class ReassignRole {

    public static void main(String[] args) throws Exception {

        final String endpointAddress = args[0];
        final String username = args[1];
        final String password = args[2];

        // Configure an authenticator to provide the credentials
        // for the web service
        Authenticator.setDefault(new Authenticator() {
```

```
            @Override
            protected PasswordAuthentication getPasswordAuthentication() {
                return new PasswordAuthentication(username,
password.toCharArray());
            }
        });

        // Get the document right operations endpoint
        DocumentRightOperationsService service = new
DocumentRightOperationsService();

        DocumentRightOperations rightOperations =
service.getDocumentRightOperations();

        // Set the end point address
        Map<String, Object> requestContext =
((BindingProvider)rightOperations).getRequestContext();

        requestContext.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
endpointAddress);

        // Reference the account by user name
        AccountRef accountRef = new AccountRef();

        accountRef.setUuid("urn:user:" + URLEncoder.encode(username, "utf-8"));

        // Get all rights assigned to the account
        List<DocumentRight> rights =
rightOperations.listRightsByAccount(accountRef);

        // Take the first one on the list
        DocumentRight right = rights.get(0);

        DocumentRightRef rightRef = new DocumentRightRef();
        rightRef.setUuid(right.getUuid());

        // Get a reference to the role to be reassigned
        DomainRef domainRef = right.getRole().getDomain();

        DocumentRoleRef roleRef = new DocumentRoleRef();

        roleRef.setUuid(right.getRole().getUuid());
        roleRef.setDomain(domainRef);

        // Change the item exclusion list to contain one sample item
        ItemCode itemCode = new ItemCode();
        itemCode.setValue("sample-item-code");

        // Reassign the role to the account
        rightOperations.reassignRole(Collections.singletonList(rightRef),
                roleRef, Collections.singletonList(itemCode));
    }
}
```

6.2.11 Sealing a File

The following code demonstrates how to seal a file. The content to seal can be provided as any type of `InputStream`; this example uses a file input stream. The sample writes the resulting stream out as a file with a sealed file name inferred from

the unsealed file name. The file is sealed using the context classification system, specifying a context with a known UUID value and an item code.

Example 6–8

```
import generated.Classification;
import generated.ClassificationSystemRef;
import generated.ContentType;
import generated.ContentTypeOperations;
import generated.ContentTypeOperationsService;
import generated.SealingOptions;
import generated.SealingServices;
import generated.SealingServicesService;

import java.io.File;
import java.io.FileOutputStream;
import java.io.StringReader;
import java.util.GregorianCalendar;
import java.util.Map;

import javax.activation.DataHandler;
import javax.activation.FileDataSource;
import javax.xml.datatype.DatatypeFactory;
import javax.xml.datatype.XMLGregorianCalendar;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.ws.BindingProvider;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.xml.sax.InputSource;

import com.sun.xml.ws.developer.JAXWSProperties;

public class SealFile {

    /**
     * MTOM threshold.
     *
     * The size in bytes that binary data should be before being sent as an
attachment in the
     * web service request or response.
     *
     * <br/><br/>Value: <tt>{@value}</tt>
     */
    static public final int MTOM_THRESHOLD = 16384;

    public static void main(String[] args) throws Exception {

        String hostPort = args[0];
        String username = args[1];
        String password = args[2];

        // The server URI. e.g. https://irm.example.com/irm_desktop
        String serverURI = args[3];

        // The filename to seal
        String filename = args[4];

        // Context UUID is fixed for sample code
```

```
        String contextUUID = "46f910d9-dd30-476e-b060-4d01f88f8b05";

        // Date for the item code time stamp
        DatatypeFactory dataTypeFactory = DatatypeFactory.newInstance();

        XMLGregorianCalendar date = dataTypeFactory.newXMLGregorianCalendar(
            new GregorianCalendar());

        // Create a context cookie for the classification - this specifies which
context to use as
        // well as the item code for the content.
        //
        // Specifies an explicit item code value and time.
        String cookieXMLText =
            "<?xml version=\"1.0\" encoding=\"UTF-8\"?>" +
            "<classifications:ContextCookie
xmlns:classifications=\"http://xmlns.oracle.com/irm/classifications\">" +
            "    <context>" +
            "        <uuid>" + contextUUID + "</uuid>" +
            "    </context>" + "    <itemCode>" +
            "        <value>" + new File(filename).getName() + "</value>" +
            "        <time>" + date.toString() + "</time>" +
            "    </itemCode>" + "</classifications:ContextCookie>";

        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        DocumentBuilder builder = factory.newDocumentBuilder();

        Document document = builder.parse(new InputSource(new StringReader(
            cookieXMLText.toString())));

        Element cookieElement = document.getDocumentElement();

        // Create the classification details used in the sealing options
        Classification classification = new Classification();

        // For the context classification system the classification Id is the
context UUID value.
        classification.setId("46f910d9-dd30-476e-b060-4d01f88f8b05");

        // Context classification system
        ClassificationSystemRef contextSystemRef = new ClassificationSystemRef();
        contextSystemRef.setUuid("37c8da32-5420-4146-816c-27f63de27250");

        classification.setSystem(contextSystemRef);

        // As the key set is not known get the sealing process to automatically
fill this in
        classification.setKeySet(null);

        // URL sealed into content that tells the desktop where to go to get
licenses
        classification.setUri(serverURI);

        // Classification time set explicitly to the current time
        classification.setClassificationTime(date);

        // Set the context and item code details
        classification.setAny(cookieElement);

        // The classification is the only mandatory property for sealing options
```

```
        SealingOptions sealingOptions = new SealingOptions();

        sealingOptions.setClassification(classification);

        // Get the content type operations web service proxy
        ContentTypeOperationsService contentTypeOperationsService =
            new ContentTypeOperationsService();

        ContentTypeOperations contentTypeOperations =
            contentTypeOperationsService.getContentTypeOperations();

        // Set the end point address for content type operations
        Map<String, Object> requestContext =
((BindingProvider)contentTypeOperations).getRequestContext();

        requestContext.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, hostPort +
"/irm_sealing/content_type_operations");

        // Set the user name and password for content type operations
        requestContext.put(BindingProvider.USERNAME_PROPERTY, username);
        requestContext.put(BindingProvider.PASSWORD_PROPERTY, password);

        // Get the MIME type of the file to seal, this is inferred from the
unsealed file name
        ContentType contentType =
contentTypeOperations.getContentTypeFromPath(filename);

        String mimeType = contentType.getMimeTypes().get(0);

        // Get the sealing services web service proxy
        SealingServicesService sealingServicesService = new
SealingServicesService();

        SealingServices sealingServices =
sealingServicesService.getSealingServices(
            new javax.xml.ws.soap.MTOMFeature(true, MTOM_THRESHOLD));

        // Set the end point address for sealing services
        requestContext = ((BindingProvider)sealingServices).getRequestContext();

        requestContext.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, hostPort +
"/irm_sealing/sealing_services");

        // Set the user name and password for sealing services
        requestContext.put(BindingProvider.USERNAME_PROPERTY, username);
        requestContext.put(BindingProvider.PASSWORD_PROPERTY, password);

        // Without this setting the client may get an
java.lang.OutOfMemoryException
        // when large files are buffered into memory by the HTTP stack.
        //
        // For more information see:
        //  https://jax-ws.dev.java.net/guide/HTTP_client_streaming_support.html
        //  https://jax-ws.dev.java.net/guide/Large_Attachments.html
        requestContext.put(JAXWSProperties.HTTP_CLIENT_STREAMING_CHUNK_SIZE,
4096);

        // Send the file contents to the server for sealing
        DataHandler input = new DataHandler(new FileDataSource(filename));
```

```
        DataHandler results = sealingServices.seal(input, mimeType,
sealingOptions);

        // Get the sealed equivalent of the unsealed filename
        String sealedFilename = contentTypeOperations.getSealedFileName(filename);

        // Write the stream out to a file
        FileOutputStream outputStream = new FileOutputStream(sealedFilename);

        results.writeTo(
        outputStream);

        // Close the streams
        outputStream.close();
    }
}
```

6.2.12 Peeking a Sealed File

The following code demonstrates how to extract the metadata from sealed content using the peek method. This method sends the sealed content to the sealing server, the server extracts the metadata and returns this information to the caller. The sealed content can be provided as any type of `InputStream`; this example uses a file input stream. Once peeked the file metadata, which includes the Classification details, can be examined. The sample code prints out the human readable classification details (the labels) that were sealed into the content.

Example 6–9

```
import generated.Classification;
import generated.ContentDescription;
import generated.Label;
import generated.SealingServices;
import generated.SealingServicesService;

import java.util.Map;

import javax.activation.DataHandler;
import javax.activation.FileDataSource;
import javax.xml.ws.BindingProvider;

import com.sun.xml.ws.developer.JAXWSProperties;

public class PeekFile {
    /**
     * MTOM threshold.
     *
     * The size in bytes that binary data should be before being sent as an
attachment in the
     * web service request or response.
     *
     * <br/><br/>Value: <tt>{@value}</tt>
     */
    static public final int MTOM_THRESHOLD = 16384;

    public static void main(String[] args) throws Exception {

        String endpointAddress = args[0];
        String username = args[1];
        String password = args[2];
```

```
        // The name of the file to peek
        String filename = args[3];

        // Get the sealing services web service proxy
        SealingServicesService sealingServicesService = new
SealingServicesService();

        SealingServices sealingServices =
sealingServicesService.getSealingServices(
            new javax.xml.ws.soap.MTOMFeature(true, MTOM_THRESHOLD));

        // Set the end point address
        Map<String, Object> requestContext =
((BindingProvider)sealingServices).getRequestContext();

        requestContext.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
endpointAddress);

        // Set the user name and password
        requestContext.put(BindingProvider.USERNAME_PROPERTY, username);
        requestContext.put(BindingProvider.PASSWORD_PROPERTY, password);

        // Without this setting the client may get an
java.lang.OutOfMemoryException
        // when large files are buffered into memory by the HTTP stack.
        //
        // For more information see:
        //  https://jax-ws.dev.java.net/guide/HTTP_client_streaming_support.html
        //  https://jax-ws.dev.java.net/guide/Large_Attachments.html
        requestContext.put(JAXWSProperties.HTTP_CLIENT_STREAMING_CHUNK_SIZE,
4096);

        // Perform the peek, providing a stream to the sealed file
        DataHandler input = new DataHandler(new FileDataSource(filename));

        ContentDescription results = sealingServices.peek(input);

        // Extract the classification details from the content
        Classification classification = results.getClassification();

        // Show all the labels sealed into content (assumes labels are available)
        for (Label label : classification.getLabels()) {
            System.out.println(label.getLocale() + " : " + label.getName());
        }
    }
}
```

6.2.13 Peeking a Sealed File and Checking the Digital Signature

The following code demonstrates how to extract the metadata from sealed content using the validatedPeek method. This method sends the sealed content to the sealing server, the server extracts the metadata and returns this information to the caller. The sealed content can be provided as any type of `InputStream`; this example uses a file input stream. Once peeked the file metadata, which includes the Classification details, can be examined. The sample code prints out the human readable classification details (the labels) that were sealed into the content.

Example 6–10

```
import generated.Classification;
import generated.ContentDescription;
import generated.Label;
import generated.SealingServices;
import generated.SealingServicesService;

import java.util.Map;

import javax.activation.DataHandler;
import javax.activation.FileDataSource;
import javax.xml.ws.BindingProvider;

import com.sun.xml.ws.developer.JAXWSProperties;

public class ValidatedPeekFile {

    /**
     * MTOM threshold.
     *
     * The size in bytes that binary data should be before being sent as an
attachment in the
     * web service request or response.
     *
     * <br/><br/>Value: <tt>{@value}</tt>
     */
    static public final int MTOM_THRESHOLD = 16384;

    public static void main(String[] args) throws Exception {

        String endpointAddress = args[0];
        String username = args[1];
        String password = args[2];

        // The name of the file to peek
        String filename = args[3];

        // Get the sealing services web service proxy
        SealingServicesService sealingServicesService = new
SealingServicesService();

        SealingServices sealingServices =
sealingServicesService.getSealingServices(
            new javax.xml.ws.soap.MTOMFeature(true, MTOM_THRESHOLD));

        // Set the end point address
        Map<String, Object> requestContext =
((BindingProvider)sealingServices).getRequestContext();

        requestContext.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
endpointAddress);

        // Set the user name and password
        requestContext.put(BindingProvider.USERNAME_PROPERTY, username);
        requestContext.put(BindingProvider.PASSWORD_PROPERTY, password);

        // Without this setting the client may get an
java.lang.OutOfMemoryException
        // when large files are buffered into memory by the HTTP stack.
        //
```

```
        // For more information see:
        //  https://jax-ws.dev.java.net/guide/HTTP_client_streaming_support.html
        //  https://jax-ws.dev.java.net/guide/Large_Attachments.html
        requestContext.put(JAXWSProperties.HTTP_CLIENT_STREAMING_CHUNK_SIZE,
4096);

        // Send the file contents to the server for peeking
        DataHandler input = new DataHandler(new FileDataSource(filename));

        ContentDescription results = sealingServices.validatedPeek(input);

        // Extract the classification details from the content
        Classification classification = results.getClassification();

        // Show all the labels sealed into content (assumes labels are available)
        for (Label label : classification.getLabels()) {
            System.out.println(label.getLocale() + " : " + label.getName());
        }
    }
}
```

6.2.14 Changing Item Restrictions Associated with a Right

The following code demonstrates how to alter the item locks or exclusions associated with a right. The sample code replaces one item code with two item codes.

Example 6–11

```
import generated.AccountRef;
import generated.DocumentRight;
import generated.DocumentRightOperations;
import generated.DocumentRightOperationsService;
import generated.DocumentRightRef;
import generated.ItemCode;

import java.net.Authenticator;
import java.net.PasswordAuthentication;
import java.net.URLEncoder;
import java.util.ArrayList;
import java.util.Collections;
import java.util.GregorianCalendar;
import java.util.List;
import java.util.Map;

import javax.xml.datatype.DatatypeFactory;
import javax.xml.ws.BindingProvider;

public class SaveChangesToItems {

    public static void main(String[] args) throws Exception {

        final String endpointAddress = args[0];
        final String username = args[1];
        final String password = args[2];

        // Configure an authenticator to provide the credentials
        // for the web service
        Authenticator.setDefault(new Authenticator() {
            @Override
            protected PasswordAuthentication getPasswordAuthentication() {
```

```
                return new PasswordAuthentication(username,
password.toCharArray());
            }
        });

        // Get the document right operations endpoint
        DocumentRightOperationsService service = new
DocumentRightOperationsService();

        DocumentRightOperations rightOperations =
service.getDocumentRightOperations();

        // Set the end point address
        Map<String, Object> requestContext =
((BindingProvider)rightOperations).getRequestContext();

        requestContext.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
endpointAddress);

        // Reference the account by user name
        AccountRef accountRef = new AccountRef();

        accountRef.setUuid("urn:user:" + URLEncoder.encode(username, "utf-8"));

        // Get all rights assigned to the account
        List<DocumentRight> rights =
rightOperations.listRightsByAccount(accountRef);

        // Take the first one on the list
        DocumentRight right = rights.get(0);

        DocumentRightRef rightRef = new DocumentRightRef();
        rightRef.setUuid(right.getUuid());

        // The save change method allows items to be added and/or removed in the
same call.
        // It does this be comparing two sets of items and applying the
differences.

        // Item codes
        ItemCode sampleItemCode = new ItemCode();
        sampleItemCode.setValue("sample-item-code");

        DatatypeFactory datatypeFactory = DatatypeFactory.newInstance();

        ItemCode sampleItemCodeOne = new ItemCode();
        sampleItemCodeOne.setValue("sample-item-code-one");
        sampleItemCodeOne.setTime(datatypeFactory.newXMLGregorianCalendar(new
GregorianCalendar()));

        ItemCode sampleItemCodeTwo = new ItemCode();
        sampleItemCodeTwo.setValue("sample-item-code-two");
        sampleItemCodeTwo.setTime(datatypeFactory.newXMLGregorianCalendar(new
GregorianCalendar()));

        // This example shows a delta where item "sample-item-code" is removed
        // and items "sample-item-code-one" and "sample-item-code-two" are added.
        List<ItemCode> itemCodes = Collections.singletonList(sampleItemCode);

        List<ItemCode> deltaItemCodes = new ArrayList<ItemCode>();
```

```
        deltaItemCodes.add(sampleItemCodeOne);
        deltaItemCodes.add(sampleItemCodeTwo);

        // Alter the items
        rightOperations.saveChangesToItems(Collections.singletonList(rightRef),
                itemCodes, deltaItemCodes);
    }
}
```

6.2.15 Unassigning Rights Assigned to a User

The following code demonstrates how to unassign rights that have been assigned to a user. The sample first lists all the rights directly assigned to the user and unassigns them. To unassign the right the authenticated user must be a context manager for the related context.

Example 6–12

```
import generated.AccountRef;
import generated.DocumentRight;
import generated.DocumentRightOperations;
import generated.DocumentRightOperationsService;
import generated.DocumentRightRef;

import java.net.Authenticator;
import java.net.PasswordAuthentication;
import java.net.URLEncoder;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;

import javax.xml.ws.BindingProvider;

public class UnassignRights {

    public static void main(String[] args) throws Exception {

        final String endpointAddress = args[0];
        final String username = args[1];
        final String password = args[2];

        // Configure an authenticator to provide the credentials
        // for the web service
        Authenticator.setDefault(new Authenticator() {
            @Override
            protected PasswordAuthentication getPasswordAuthentication() {
                return new PasswordAuthentication(username,
password.toCharArray());
            }
        });

        // Get the document right operations endpoint
        DocumentRightOperationsService service = new
DocumentRightOperationsService();

        DocumentRightOperations rightOperations =
service.getDocumentRightOperations();

        // Set the end point address
```

```
        Map<String, Object> requestContext =
((BindingProvider)rightOperations).getRequestContext();

        requestContext.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
endpointAddress);

        // Reference the account by user name
        AccountRef accountRef = new AccountRef();

        accountRef.setUuid("urn:user:" + URLEncoder.encode(username, "utf-8"));

        // Get all rights assigned to the account
        List<DocumentRight> rights =
rightOperations.listRightsByAccount(accountRef);

        List<DocumentRightRef> rightRefs = new
ArrayList<DocumentRightRef>(rights.size());

        for (DocumentRight right : rights) {
            DocumentRightRef rightRef = new DocumentRightRef();
            rightRef.setUuid(right.getUuid());

            rightRefs.add(rightRef);
        }

        // Unassign the rights
        rightOperations.unassignRights(rightRefs);
    }
}
```

6.2.16 Reclassifying a File

The following code demonstrates how to reclassify a sealed file using the reclassify method. The content to reclassify can be provided as any type of `InputStream`; this example uses a file input stream. The sample changes the labels of the classification and then writes the resulting stream out as a file.

Example 6–13

```
import generated.Classification;
import generated.ContentDescription;
import generated.Label;
import generated.SealingServices;
import generated.SealingServicesService;

import java.io.FileOutputStream;
import java.util.Locale;
import java.util.Map;

import javax.activation.DataHandler;
import javax.activation.FileDataSource;
import javax.xml.ws.BindingProvider;

import com.sun.xml.ws.developer.JAXWSProperties;

public class ReclassifyFile {

    /**
     * MTOM threshold.
     *
```

```
     * The size in bytes that binary data should be before being sent as an
attachment in the
     * web service request or response.
     *
     * <br/><br/>Value: <tt>{@value}</tt>
     */
    static public final int MTOM_THRESHOLD = 16384;

public static void main(String[] args) throws Exception {

    String endpointAddress = args[0];
        String username = args[1];
        String password = args[2];

        // Get the file to reclassify
        String filename = args[3];

        // Get the label to apply to the classification
        String labelName = args[4];

        // Get the sealing services web service proxy
        SealingServicesService sealingServicesService = new
SealingServicesService();

        SealingServices sealingServices =
sealingServicesService.getSealingServices(
            new javax.xml.ws.soap.MTOMFeature(true, MTOM_THRESHOLD));

        // Set the end point address
        Map<String, Object> requestContext =
((BindingProvider)sealingServices).getRequestContext();

        requestContext.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
endpointAddress);

        // Set the user name and password
        requestContext.put(BindingProvider.USERNAME_PROPERTY, username);
        requestContext.put(BindingProvider.PASSWORD_PROPERTY, password);

        // Without this setting the client may get an
java.lang.OutOfMemoryException
        // when large files are buffered into memory by the HTTP stack.
        //
        // For more information see:
        //  https://jax-ws.dev.java.net/guide/HTTP_client_streaming_support.html
        //  https://jax-ws.dev.java.net/guide/Large_Attachments.html
        requestContext.put(JAXWSProperties.HTTP_CLIENT_STREAMING_CHUNK_SIZE,
4096);

        // Peek the contents of the file to obtain the classification details
        DataHandler input = new DataHandler(new FileDataSource(filename));

        ContentDescription contentDescription = sealingServices.peek(input);

        // Extract the classification from the content description
        Classification classification = contentDescription.getClassification();

        // Replace the labels with one
        Label label = new Label();
```

```
        label.setLocale(Locale.ENGLISH.toString());
        label.setName(labelName);

        classification.getLabels().add(label);

        // Reclassify the sealed file with the new classification
        DataHandler results = sealingServices.reclassify(input,classification);

        // Write the stream out to the same file
        FileOutputStream outputStream = new FileOutputStream(filename);

        results.writeTo(
        outputStream);

        // Close the streams
        outputStream.close();
}
}
```

6.2.17 Resealing a File with Different Custom Data

The following code demonstrates how to reseal a sealed file using the reseal method. The content to reseal can be provided as any type of `InputStream`; this example uses a file input stream. The sample adds XML-based custom data to the sealed file.

Example 6–14

```
import generated.CustomData;
import generated.SealingServices;
import generated.SealingServicesService;

import java.io.FileOutputStream;
import java.util.Collections;
import java.util.Map;
import java.util.UUID;

import javax.activation.DataHandler;
import javax.activation.FileDataSource;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.ws.BindingProvider;

import org.w3c.dom.Document;
import org.w3c.dom.Element;

import com.sun.xml.ws.developer.JAXWSProperties;

public class ResealFile {

    /**
     * MTOM threshold.
     *
     * The size in bytes that binary data should be before being sent as an
attachment in the
     * web service request or response.
     *
     * <br/><br/>Value: <tt>{@value}</tt>
     */
    static public final int MTOM_THRESHOLD = 16384;
```

```
public static void main(String[] args) throws Exception {

    String endpointAddress = args[0];
        String username = args[1];
        String password = args[2];

        // Get the file to reseal
        String filename = args[3];

        // Get the sealing services web service proxy
        SealingServicesService sealingServicesService = new
SealingServicesService();

        SealingServices sealingServices =
sealingServicesService.getSealingServices(
            new javax.xml.ws.soap.MTOMFeature(true, MTOM_THRESHOLD));

        // Set the end point address
        Map<String, Object> requestContext =
((BindingProvider)sealingServices).getRequestContext();

        requestContext.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
endpointAddress);

        // Set the user name and password
        requestContext.put(BindingProvider.USERNAME_PROPERTY, username);
        requestContext.put(BindingProvider.PASSWORD_PROPERTY, password);

        // Without this setting the client may get an
java.lang.OutOfMemoryException
        // when large files are buffered into memory by the HTTP stack.
        //
        // For more information see:
        //  https://jax-ws.dev.java.net/guide/HTTP_client_streaming_support.html
        //  https://jax-ws.dev.java.net/guide/Large_Attachments.html
        requestContext.put(JAXWSProperties.HTTP_CLIENT_STREAMING_CHUNK_SIZE,
4096);

        // Custom data is provided as XML
        DocumentBuilderFactory documentBuilderFactory =
DocumentBuilderFactory.newInstance();
        DocumentBuilder documentBuilder =
documentBuilderFactory.newDocumentBuilder();
        Document document = documentBuilder.newDocument();

        Element element = document.createElement("SampleCustomData");
        element.setTextContent("Some example custom data provided as an XML text
element");

        CustomData data = new CustomData();

        // UUID identifies the custom data, in this example just use a random UUID
value
        data.setUuid(UUID.randomUUID().toString());

        // Custom data is XML document
        data.setAny(element);

        // Send the sealed file contents to the server for resealing
        DataHandler input = new DataHandler(new FileDataSource(filename));
```

```
        DataHandler results = sealingServices.reseal(input,
Collections.singletonList(data));

        // Write the stream out to a file
        FileOutputStream outputStream = new FileOutputStream(filename);

        results.writeTo(
        outputStream);

        // Close the streams
        outputStream.close();
}
}
```

6.2.18 Unsealing a File

The following code demonstrates how to unseal a sealed file using the unseal method. The content to unseal can be provided as any type of `InputStream`; this example uses a file input stream. The sample writes the resulting unsealed stream out to a file.

Example 6–15

```
import generated.SealingServices;
import generated.SealingServicesService;

import java.io.FileOutputStream;
import java.util.Map;

import javax.activation.DataHandler;
import javax.activation.FileDataSource;
import javax.xml.ws.BindingProvider;

import com.sun.xml.ws.developer.JAXWSProperties;

public class UnsealFile {

    /**
     * MTOM threshold.
     *
     * The size in bytes that binary data should be before being sent as an
attachment in the
     * web service request or response.
     *
     * <br/><br/>Value: <tt>{@value}</tt>
     */
    static public final int MTOM_THRESHOLD = 16384;

    public static void main(String[] args) throws Exception {

        String endpointAddress = args[0];
        String username = args[1];
        String password = args[2];

        // The file to unseal
        String sealedFilename = args[3];

        // The unsealed file name
        String unsealedFilename = args[4];
```

```
        // Get the sealing services web service proxy
        SealingServicesService sealingServicesService = new
SealingServicesService();

        SealingServices sealingServices =
sealingServicesService.getSealingServices(
            new javax.xml.ws.soap.MTOMFeature(true, MTOM_THRESHOLD));

        // Set the end point address
        Map<String, Object> requestContext =
((BindingProvider)sealingServices).getRequestContext();

        requestContext.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
endpointAddress);

        // Set the user name and password
        requestContext.put(BindingProvider.USERNAME_PROPERTY, username);
        requestContext.put(BindingProvider.PASSWORD_PROPERTY, password);

        // Without this setting the client may get an
java.lang.OutOfMemoryException
        // when large files are buffered into memory by the HTTP stack.
        //
        // For more information see:
        //  https://jax-ws.dev.java.net/guide/HTTP_client_streaming_support.html
        //  https://jax-ws.dev.java.net/guide/Large_Attachments.html
        requestContext.put(JAXWSProperties.HTTP_CLIENT_STREAMING_CHUNK_SIZE,
4096);

        // Send the file contents to the server for unsealing
        DataHandler input = new DataHandler(new FileDataSource(sealedFilename));

        DataHandler results = sealingServices.unseal(input);

        // Write the stream out to a file
        FileOutputStream outputStream = new FileOutputStream(unsealedFilename);

        results.writeTo(
        outputStream);

        // Close the streams
        outputStream.close();
    }
}
```

6.2.19 Listing Classifications

The following code demonstrates how to list classification details from a sealing server using the listClassifications method. The sample code displays the list of Classifications details available to the authenticated user.

Example 6–16

```
import generated.Classification;
import generated.DesktopServices;
import generated.DesktopServicesService;
import generated.Label;

import java.net.Authenticator;
import java.net.PasswordAuthentication;
```

```
import java.util.Collection;
import java.util.Map;

import javax.xml.ws.BindingProvider;

public class ListClassifications {

    public static void main(String[] args) throws Exception {

        final String endpointAddress = args[0];
        final String username = args[1];
        final String password = args[2];

        // Configure an authenticator to provide the credentials
        // for the web service
        Authenticator.setDefault(new Authenticator() {
            @Override
            protected PasswordAuthentication getPasswordAuthentication() {
                return new PasswordAuthentication(username,
password.toCharArray());
            }
        });

        // Get the desktop services web service
        DesktopServicesService desktopServicesService = new
DesktopServicesService();

        DesktopServices desktopServices =
desktopServicesService.getDesktopServices();

        // Set the end point address
        Map<String, Object> requestContext =
((BindingProvider)desktopServices).getRequestContext();

        requestContext.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
endpointAddress);

        // The server URI. e.g. https://irm.example.com/irm_desktop
        String serverURI = args[3];

        // List classifications available to the account
        Collection<Classification> classifications =
desktopServices.listClassifications(serverURI);

        // Display the labels of the classifications
        for (Classification classification : classifications) {

            for (Label label : classification.getLabels()) {
                System.out.println(label.getLocale() + " : " + label.getName());
            }
        }
    }
}
```

6.2.20 Searching the Context Journal Using Web Services

The following code demonstrates how to search the content usage for context classified content. The sample code searches for all entries for the last twenty-four hours and displays a short summary for the first one hundred entries.

Example 6–17

```
import generated.ContextJournalEntry;
import generated.ContextOperations;
import generated.ContextOperationsService;
import generated.PageRange;
import generated.TimeRange;

import java.net.Authenticator;
import java.net.PasswordAuthentication;
import java.util.Calendar;
import java.util.GregorianCalendar;
import java.util.List;
import java.util.Map;

import javax.xml.datatype.DatatypeFactory;
import javax.xml.datatype.XMLGregorianCalendar;
import javax.xml.ws.BindingProvider;

public class SearchJournal {

    public static void main(String[] args) throws Exception {

        // The server address. e.g. https://irm.example.com
        final String endpointAddress = args[0];
        final String username = args[1];
        final String password = args[2];

        // Configure an authenticator to provide the credentials
        // for the web service
        Authenticator.setDefault(new Authenticator() {
            @Override
            protected PasswordAuthentication getPasswordAuthentication() {
                return new PasswordAuthentication(username,
password.toCharArray());
            }
        });

        // Get the content operations endpoint
        ContextOperationsService service = new ContextOperationsService();

        ContextOperations contextOperations = service.getContextOperations();

        // Set the end point address
        Map<String, Object> requestContext =
((BindingProvider)contextOperations).getRequestContext();

        requestContext.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
endpointAddress);

        // Use a calendar to work out the time range
        GregorianCalendar calendar = new GregorianCalendar();

        DatatypeFactory datatypeFactory = DatatypeFactory.newInstance();

        // Search for all records from the last 24 hours
        XMLGregorianCalendar end =
datatypeFactory.newXMLGregorianCalendar(calendar);

        calendar.add(Calendar.DAY_OF_MONTH, -1);
```

```
        XMLGregorianCalendar begin =
datatypeFactory.newXMLGregorianCalendar(calendar);

        TimeRange timeRange = new TimeRange();

        timeRange.setBegin(begin);
        timeRange.setEnd(end);

        // Search the context journal
        PageRange pageRange = new PageRange();

        pageRange.setFirst(1);
        pageRange.setLast(100);

        List<ContextJournalEntry> journalResults =
contextOperations.searchJournal(
            null, // no accounts filter
            null, // no item codes filter
            timeRange,
            pageRange,
            null); // no sorting details

        if (journalResults.size() == 0)
            return;

        // Display the timestamp, URI and and feature for each entry
        for (ContextJournalEntry entry : journalResults) {

            System.out.print("Timestamp : " + entry.getTime());
            System.out.print(" Account   : " + entry.getAccount().getName());
            System.out.print(" Content   : " + (entry.getUri() != null ?
entry.getUri() : ""));
            System.out.print(" Feature   : " + entry.getFeature().getId());
        }
    }
}
```

6.2.21 Checking in Licenses

The following code demonstrates how to check in licenses currently checked out to a sealing server. When the sealing server processes content it will usually check out licenses for the authenticated user. These licenses can no longer be used from other locations (for example, the Oracle IRM Desktop) until they expire or are manually checked in.

Example 6–18

```
import generated.DesktopServices;
import generated.DesktopServicesService;

import java.net.Authenticator;
import java.net.PasswordAuthentication;
import java.util.Map;

import javax.xml.ws.BindingProvider;

public class CheckIn {

    public static void main(String[] args) {
```

```
        final String endpointAddress = args[0];
        final String username = args[1];
        final String password = args[2];

        // Configure an authenticator to provide the credentials
        // for the web service
        Authenticator.setDefault(new Authenticator() {
            @Override
            protected PasswordAuthentication getPasswordAuthentication() {
                return new PasswordAuthentication(username,
password.toCharArray());
            }
        });

        // Get the desktop services web service
        DesktopServicesService desktopServicesService = new
DesktopServicesService();

        DesktopServices desktopServices =
desktopServicesService.getDesktopServices();

        // Set the end point address
        Map<String, Object> requestContext =
((BindingProvider)desktopServices).getRequestContext();

        requestContext.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
endpointAddress);

        // The server URI. e.g. https://irm.example.com/irm_desktop
        String serverURI = args[3];

        // Check in all the licenses currently within the
        // desktop store for the given server
        desktopServices.checkIn(serverURI);
    }
}
```

6.2.22 Deleting a Domain

The following code demonstrates how to delete a domain. The sample code uses a fixed domain UUID for the new domain so that all sample code can work with a known domain. A new domain would typically be given a new random UUID value. The authenticated user must be a domain administrator. When a domain is deleted all the associated roles, templates and contexts are also deleted.

Example 6–19

```
import generated.DomainOperations;
import generated.DomainOperationsService;
import generated.DomainRef;

import java.net.Authenticator;
import java.net.PasswordAuthentication;
import java.util.Map;

import javax.xml.ws.BindingProvider;

public class DeleteDomain {

    public static void main(String[] args) throws Exception {
```

```
        final String endpointAddress = args[0];
        final String username = args[1];
        final String password = args[2];

        // Configure an authenticator to provide the credentials
        // for the web service
        Authenticator.setDefault(new Authenticator() {
            @Override
            protected PasswordAuthentication getPasswordAuthentication() {
                return new PasswordAuthentication(username,
password.toCharArray());
            }
        });

        // Get the domain operations web service
        DomainOperationsService service = new DomainOperationsService();

        DomainOperations domainOperations = service.getDomainOperations();

        // Set the end point address
        Map<String, Object> requestContext =
((BindingProvider)domainOperations).getRequestContext();

        requestContext.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
endpointAddress);

        // Domain UUID is fixed for sample code
        DomainRef domain = new DomainRef();

        domain.setUuid("6fab93fd-2858-461a-a0b3-34e261dbf8fd");

        // Delete the domain using the domain reference
        domainOperations.deleteDomain(domain);
    }
}
```

6.3 Using the Oracle IRM Web Service Code

This section contains the following topics:

- Introduction
- Class Path
- Differences from the JDeveloper Generated Code
- Creating a Domain
- Creating a Role
- Creating a Context Template
- Creating a Context
- Assigning a Role to a User
- Listing Rights Assigned to a User or Group
- Altering the Role Assigned to a User or Group
- Sealing a File
- Peeking a Sealed File

- Peeking a Sealed File and Checking the Digital Signature
- Changing Item Restrictions Associated with a Right
- Unassigning Rights Assigned to a User
- Reclassifying a File
- Resealing a File with Different Custom Data
- Unsealing a File
- Listing Classifications
- Searching the Context Journal Using Web Services
- Checking in Licenses
- Deleting a Domain

6.3.1 Introduction

The following section provides example code that can be used with IRM provided JAX-WS web service proxies.

6.3.2 Class Path

The code samples in this section require the following jar files to be added to the class path. These jar files contain the web service proxy code and objects equivalent to those that would be generated with a web service proxy generated.

- `irm-common.jar`
- `irm-engine.jar`
- `irm-ws.jar`

These jar files also provide the WSDL and XSD files required to use the web service proxies.

> **Note:** These jar files are not required if other WSDL web service proxy code generators are used (such as the JDeveloper web service proxy generator).

6.3.3 Differences from the JDeveloper Generated Code

The Oracle IRM provided web service proxies are functionality identical to the ones generated by JDeveloper (or any other web service proxy generator). However there are a few code differences:

- UUID types use `java.util.UUID` rather than `java.lang.String`.
- Date types use `java.util.Date` rather than `javax.xml.datatype.XMLGregorianCalendar`.
- The classification cookie can be provided as a `ContextCookie` object rather than an XML document.
- The classification cookie property modifier is called `setCookie` rather than `setAny`.
- Objects that represent XML types have a constructor that allows all the properties to be provided on construction.

- Constant values, such as the context classification UUID are available as static final variables.

6.3.4 Creating a Domain

The following code demonstrates how to create a domain. The sample code uses a fixed domain UUID so that all sample code can work against a known domain. A new domain would typically be given a new random UUID value. The authenticated user becomes the domain administrator. When a domain is created, a set of human-readable labels can be given to the domain for the target language audience.

Example 6–20

```
import static
oracle.irm.j2ee.jws.rights.context.DomainOperations.getDomainOperationsEndpoint;

 import java.util.Locale;
 import java.util.UUID;

 import oracle.irm.engine.types.core.general.Label;
 import oracle.irm.engine.types.rights.context.Domain;
 import oracle.irm.j2ee.jws.rights.context.DomainOperationsEndpoint;

 public class SaveNewDomainWS {

     public static void main(String[] args) throws Exception {

         String hostPort = args[0];
         String username = args[1];
         String password = args[2];

         // Get the domain operations web service
         DomainOperationsEndpoint domainOperations =
getDomainOperationsEndpoint(hostPort, username, password);

         // Domain has one English label
         Label label = new Label(Locale.ENGLISH, "Sample Domain", "This is a
domain created from sample code.");

         // Domain UUID is fixed for sample code
         UUID domainUUID =
UUID.fromString("6fab93fd-2858-461a-a0b3-34e261dbf8fd");

         Domain domain = new Domain(domainUUID,new Label[] { label });

         // Save the new domain
         domainOperations.saveNewDomain(domain);
     }
 }
```

6.3.5 Creating a Role

The following code demonstrates how to create a role. The sample code uses a fixed role UUID so that all sample code can work with a known role. A new role would typically be given a new random UUID value. The sample role is set up to allow all the content operations required by the sample code. When assigned to a user, this role allows sealing, unsealing, resealing and (validated) peeking. This is done by a providing an appropriate set of features and export constraints.

Example 6–21

```
import static oracle.irm.engine.core.feature.FeatureConstants.OPEN_FEATURE_ID;
 import static oracle.irm.engine.core.feature.FeatureConstants.RESEAL_FEATURE_ID;
 import static oracle.irm.engine.core.feature.FeatureConstants.SEAL_FEATURE_ID;
 import static
oracle.irm.j2ee.jws.rights.context.DocumentRoleOperations.getDocumentRoleOperation
sEndpoint;

 import java.util.Locale;
 import java.util.UUID;

 import oracle.irm.engine.types.classifications.item.ItemConstraints;
 import oracle.irm.engine.types.core.feature.Feature;
 import oracle.irm.engine.types.core.general.Label;
 import oracle.irm.engine.types.core.license.LicenseCriteria;
 import oracle.irm.engine.types.core.time.TimePeriod;
 import oracle.irm.engine.types.rights.context.DocumentRole;
 import oracle.irm.engine.types.rights.context.DomainRef;
 import oracle.irm.j2ee.jws.rights.context.DocumentRoleOperationsEndpoint;

 public class SaveNewRoleWS {

     public static void main(String[] args) throws Exception {

         String hostPort = args[0];
         String username = args[1];
         String password = args[2];

         // Document Role UUID is fixed for sample code
         UUID documentRoleUUID =
UUID.fromString("ee82c3f9-152b-440d-afd7-dbf36b0c8188");

         DocumentRole role = new DocumentRole();

         // The UUID value that identifies this role within the domain
         role.setUuid(documentRoleUUID);

         // Role has one English label
         Label label = new Label(Locale.ENGLISH, "Sample Role", "This is a role
created from sample code.");

         // The human readable labels
         role.setLabels( new Label[] { label } );

         // This role allows the user to access content while offline by
persisting licenses on the desktop
         role.setStorage(LicenseCriteria.Storage.PERSISTENT);

         // This role allows content to be saved in the clear (unsealing and
copying)
         role.setExportConstraints(DocumentRole.ExportConstraints.NONE);

         // This role allows opening, sealing, resealing
         Feature open = new Feature(OPEN_FEATURE_ID, Feature.Use.IMMEDIATE,
false);
         Feature seal = new Feature(SEAL_FEATURE_ID, Feature.Use.IMMEDIATE,
false);
         Feature reseal = new Feature(RESEAL_FEATURE_ID, Feature.Use.IMMEDIATE,
false);
```

```
        role.setFeatures( new Feature[] { open, seal, reseal });

        // Role allows document exclusions to be listed, by default all items are
allowed
        role.setItemConstraints(ItemConstraints.Type.EXCLUSIONS);

        // This role allows content to be opened for one hour before refreshing
the rights from the server
        TimePeriod value = new TimePeriod(1, TimePeriod.Units.HOURS);

        role.setRefreshPeriod(value);

        // This role has no additional time constraints
        role.setTimeSpans(null);

        // Get the document role operations web service
        DocumentRoleOperationsEndpoint roleOperations =
getDocumentRoleOperationsEndpoint(hostPort, username, password);

        // Domain UUID is fixed for sample code
        UUID domainUUID =
UUID.fromString("6fab93fd-2858-461a-a0b3-34e261dbf8fd");

        DomainRef domain = new DomainRef(domainUUID);

        // Save the new role
        roleOperations.saveNewRole(domain, role);
    }
 }
```

6.3.6 Creating a Context Template

The following code demonstrates how to create a context template. The sample code uses a fixed template UUID so that all sample code can work with a known template. A new template would typically be given a new random UUID value. The sample template has one role and is active. This template is used to create contexts in the create context code sample.

Example 6–22

```
import static
oracle.irm.j2ee.jws.rights.context.ContextTemplateOperations.getContextTemplateOpe
rationsEndpoint;

 import java.util.Locale;
 import java.util.UUID;

 import oracle.irm.engine.types.core.general.Label;
 import oracle.irm.engine.types.rights.context.ContextTemplate;
 import oracle.irm.engine.types.rights.context.DocumentRoleRef;
 import oracle.irm.engine.types.rights.context.DomainRef;
 import oracle.irm.j2ee.jws.rights.context.ContextTemplateOperationsEndpoint;

 public class SaveNewContextTemplateWS {

     public static void main(String[] args) throws Exception {

         String hostPort = args[0];
         String username = args[1];
         String password = args[2];
```

```
        // Context Template UUID is fixed for sample code
        UUID contextTemplateUUID =
UUID.fromString("930876e6-a505-4a10-8d93-bc43d9a37c23");

        ContextTemplate template = new ContextTemplate();

        // The UUID value that identifies this role within the domain
        template.setUuid(contextTemplateUUID);

        // Context Template has one English label
        Label label = new Label(Locale.ENGLISH, "Sample Template", "This is a
template created from sample code.");

        // The human readable labels
        template.setLabels( new Label[] { label } );

        // The template is active
        template.setStatus(ContextTemplate.Status.ACTIVE);

        // Domain UUID is fixed for sample code
        UUID domainUUID =
UUID.fromString("6fab93fd-2858-461a-a0b3-34e261dbf8fd");

        DomainRef domain = new DomainRef(domainUUID);

        // Document Role UUID is fixed for sample code
        UUID documentRoleUUID =
UUID.fromString("ee82c3f9-152b-440d-afd7-dbf36b0c8188");

        DocumentRoleRef documentRole = new DocumentRoleRef(documentRoleUUID,
domain);

        // Template has one role
        template.setRoles( new DocumentRoleRef[] { documentRole });

        // Get the context template operations web service
        ContextTemplateOperationsEndpoint templateOperations =
getContextTemplateOperationsEndpoint(hostPort, username, password);

        // Save the new template
        templateOperations.saveNewContextTemplate(domain, template);
    }
 }
```

6.3.7 Creating a Context

The following code demonstrates how to create a context from a context template. The sample code uses a fixed context template reference (information that identifies the template) and provides a fixed UUID value for the new context. The authenticated user becomes the context manager. The context is created with two labels, English and German. This context is used in the sample code that assigns a role, as well as the sealing, unsealing, resealing, reclassification and peeking code samples.

Example 6–23

```
import static
oracle.irm.j2ee.jws.rights.context.ContextOperations.getContextOperationsEndpoint;

 import java.util.Locale;
```

```
import java.util.UUID;

import oracle.irm.engine.types.core.general.Label;
import oracle.irm.engine.types.rights.context.ContextTemplateRef;
import oracle.irm.engine.types.rights.context.DomainRef;
import oracle.irm.engine.types.rights.context.ContextInstance.Visibility;
import oracle.irm.j2ee.jws.rights.context.ContextOperationsEndpoint;

public class CreateContextFromTemplateWS {

    public static void main(String[] args) throws Exception {

        String hostPort = args[0];
        String username = args[1];
        String password = args[2];

        // Domain UUID is fixed for sample code
        UUID domainUUID =
UUID.fromString("6fab93fd-2858-461a-a0b3-34e261dbf8fd");

        // Context Template UUID is for the "standard" template automatically
installed with a domain
        UUID templateUUID =
UUID.fromString("930876e6-a505-4a10-8d93-bc43d9a37c23");

        // Context UUID is fixed for sample code
        UUID contextUUID =
UUID.fromString("46f910d9-dd30-476e-b060-4d01f88f8b05");

        // Use the first domain available
        DomainRef domainRef = new DomainRef(domainUUID);

        // Use the first template available
        ContextTemplateRef templateRef = new ContextTemplateRef(templateUUID,
domainRef);

        // Get the context operations web service
        ContextOperationsEndpoint contextOperations =
getContextOperationsEndpoint(hostPort, username, password);

        // Context has two labels, English and German
        Label english = new Label(Locale.ENGLISH, "Sample Classification",
"Created from sample code.");
        Label german = new Label(Locale.GERMAN, "Beispielklassifikation",
"Verursacht vom Beispielcode.");

        // Create a context based on that template
        contextOperations.createContextFromTemplate(
            contextUUID, // context UUID value
            templateRef, // context template
            new Label[] { english, german }, // labels
            Visibility.DOMAIN, // visibility
            null); // additional context managers
    }
}
```

6.3.8 Assigning a Role to a User

The following code demonstrates how to assign a role to a user. To assign a role, the role, context and user or group must be specified. If the role is restricted to individual items then items can also be specified as in the assign role method.

Example 6–24

```
import static
oracle.irm.j2ee.jws.rights.context.DocumentRightOperations.getDocumentRightOperati
onsEndpoint;

 import java.net.URLEncoder;
 import java.util.UUID;

 import oracle.irm.engine.types.core.account.AccountRef;
 import oracle.irm.engine.types.rights.context.ContextInstanceRef;
 import oracle.irm.engine.types.rights.context.DocumentRoleRef;
 import oracle.irm.engine.types.rights.context.DomainRef;
 import oracle.irm.j2ee.jws.rights.context.DocumentRightOperationsEndpoint;

 public class AssignRoleWS {

     public static void main(String[] args) throws Exception {

         String hostPort = args[0];
         String username = args[1];
         String password = args[2];

         // Domain UUID is fixed for sample code
         UUID domainUUID =
UUID.fromString("6fab93fd-2858-461a-a0b3-34e261dbf8fd");

         DomainRef domainRef = new DomainRef(domainUUID);

         // Document Role UUID is for the "Sample Role" role
         UUID documentRoleUUID =
UUID.fromString("ee82c3f9-152b-440d-afd7-dbf36b0c8188");

         DocumentRoleRef roleRef = new DocumentRoleRef(documentRoleUUID,
domainRef);

         // Context UUID is fixed for sample code
         UUID contextUUID =
UUID.fromString("46f910d9-dd30-476e-b060-4d01f88f8b05");

         ContextInstanceRef contextInstanceRef = new
ContextInstanceRef(contextUUID);

         // Get the document right operations endpoint
         DocumentRightOperationsEndpoint rightOperations =
getDocumentRightOperationsEndpoint(hostPort, username, password);

         // Reference the account by user name
         AccountRef accountRef = new AccountRef("urn:user:" +
URLEncoder.encode(username, "utf-8"));

         // Assign the role to the account
         rightOperations.assignRole(
             contextInstanceRef,
```

```
            roleRef,
            new AccountRef[] { accountRef },
            null); // no item constraints
    }
}
```

6.3.9 Listing Rights Assigned to a User or Group

The following code demonstrates how to list the rights that have been assigned to a user or group. The code displays the role label and the context UUID from each right.

Example 6–25

```
import static
oracle.irm.j2ee.jws.rights.context.DocumentRightOperations.getDocumentRightOperati
onsEndpoint;

 import java.net.URLEncoder;

 import oracle.irm.engine.types.classifications.item.ItemCode;
 import oracle.irm.engine.types.core.account.AccountRef;
 import oracle.irm.engine.types.rights.context.DocumentRight;
 import oracle.irm.j2ee.jws.rights.context.DocumentRightOperationsEndpoint;

 public class ListRightsByAccountWS {

     public static void main(String[] args) throws Exception {

         String hostPort = args[0];
         String username = args[1];
         String password = args[2];

         // Get the document right operations web service
         DocumentRightOperationsEndpoint rightOperations =
getDocumentRightOperationsEndpoint(hostPort, username, password);

         // Reference the account by user name, allowed formats are
         //  urn:user:xxxx
         //  urn:group:xxxx
         //  00000000-0000-0000-0000-000000000000
         AccountRef accountRef = new AccountRef("urn:user:" +
URLEncoder.encode(username, "utf-8"));

         // Get all of the rights assigned to the account
         DocumentRight[] rights = rightOperations.listRightsByAccount(accountRef);

         // Display a summary of each right
         for (DocumentRight right : rights) {
             System.out.println("Account: " + right.getAccount().getUuid());
             System.out.println(" Context: " + right.getContext().getUuid());
             System.out.println(" Role: " + right.getRole().getUuid());

             // Show items
             ItemCode[] itemCodes = right.getItemCodes();

             if (itemCodes != null) {
                 for (ItemCode itemCode : itemCodes) {
                     System.out.println(" ItemCode: " + itemCode.getValue());
                 }
             }
```

```
        }
    }
}
```

6.3.10 Altering the Role Assigned to a User or Group

The following code demonstrates how to alter a role assignment using the reassignRole method over web services. The sample code adds an item code exclusion to a role assignment. Typically this method is used to alter the role, but as the sample code only has one demonstration role it shows how to alter the item restrictions.

Example 6–26

```
import static
oracle.irm.j2ee.jws.rights.context.DocumentRightOperations.getDocumentRightOperati
onsEndpoint;

 import java.net.URLEncoder;

 import oracle.irm.engine.types.classifications.item.ItemCode;
 import oracle.irm.engine.types.core.account.AccountRef;
 import oracle.irm.engine.types.rights.context.DocumentRight;
 import oracle.irm.engine.types.rights.context.DocumentRightRef;
 import oracle.irm.engine.types.rights.context.DocumentRoleRef;
 import oracle.irm.engine.types.rights.context.DomainRef;
 import oracle.irm.j2ee.jws.rights.context.DocumentRightOperationsEndpoint;

 public class ReassignRoleWS {

     public static void main(String[] args) throws Exception {

         String hostPort = args[0];
         String username = args[1];
         String password = args[2];

         // Get the document right operations web service
         DocumentRightOperationsEndpoint rightOperations =
getDocumentRightOperationsEndpoint(hostPort, username, password);

         // Reference the account by user name
         AccountRef accountRef = new AccountRef("urn:user:" +
URLEncoder.encode(username, "utf-8"));

         // Get all rights assigned to the account
         DocumentRight[] rights = rightOperations.listRightsByAccount(accountRef);

         // Take the first one on the list
         DocumentRight right = rights[0];

         DocumentRightRef rightRef = new DocumentRightRef(right.getUuid());

         // Get a reference to the role to be reassigned
         DomainRef domainRef = right.getRole().getDomain();

         DocumentRoleRef roleRef = new DocumentRoleRef(right.getRole().getUuid(),
domainRef);

         // Change the item exclusion list to contain one sample item
         ItemCode itemCode = new ItemCode();
         itemCode.setValue("sample-item-code");
```

```
                // Reassign the role to the account
                rightOperations.reassignRole(new DocumentRightRef[] { rightRef },
            roleRef, new ItemCode[] { itemCode });
            }
        }
```

6.3.11 Sealing a File

The following code demonstrates how to seal a file. The content to seal can be provided as any type of `InputStream`; this example uses a file input stream. The sample writes the resulting stream out as a file with a sealed file name inferred from the unsealed file name. The file is sealed using the context classification system, specifying a context with a known UUID value and an item code.

Example 6–27

```
import static oracle.irm.engine.classifications.context.ContextConstants.CONTEXT_CLASSIFICATION_
SYSTEM_UUID;
 import static oracle.irm.engine.content.type.ContentTypeOperationsInstance.getContentTypeFromPath;
 import static oracle.irm.engine.content.type.ContentTypeOperationsInstance.getSealedFileName;
 import static oracle.irm.engine.core.classification.ClassificationConstants.UNSPECIFIED_KEY_SET_
UUID;
 import static oracle.irm.j2ee.jws.content.sealing.SealingServices.getSealingServicesEndpoint;

 import java.io.File;
 import java.io.FileInputStream;
 import java.io.FileOutputStream;
 import java.io.InputStream;
 import java.net.URI;
 import java.util.Date;
 import java.util.UUID;

 import oracle.irm.engine.content.type.ContentType;
 import oracle.irm.engine.types.classifications.context.ContextCookie;
 import oracle.irm.engine.types.classifications.context.ContextRef;
 import oracle.irm.engine.types.classifications.item.ItemCode;
 import oracle.irm.engine.types.content.key.KeySetRef;
 import oracle.irm.engine.types.content.sealing.SealingOptions;
 import oracle.irm.engine.types.core.classification.Classification;
 import oracle.irm.engine.types.core.classification.ClassificationSystemRef;
 import oracle.irm.j2ee.jws.content.sealing.SealingServicesEndpoint;

 public class SealFile {

     public static void main(String[] args) throws Exception {

         String hostPort = args[0];
         String username = args[1];
         String password = args[2];

         // The server URI. for example https://irm.example.com/irm_desktop
         URI serverURI = URI.create(args[3]);

         // The filename to seal
         String unsealedFilename = args[4];

         // Context UUID is fixed for sample code
         ContextRef context = new
ContextRef(UUID.fromString("46f910d9-dd30-476e-b060-4d01f88f8b05"));
```

```
        // Provide an explicit item code value and time
        ItemCode itemCode = new ItemCode();
        itemCode.setValue(new File(unsealedFilename).getName());
        itemCode.setTime(new Date());

        // Create a context cookie for the classification - this specifies which context to use as
        // well as the item code for the content.
        ContextCookie cookie = new ContextCookie();
        cookie.setContext(context);
        cookie.setItemCode(itemCode);

        // Create the classification details used in the sealing options
        Classification classification = new Classification();

        // For the context classification system the classification Id is the context UUID value.
        classification.setId("46f910d9-dd30-476e-b060-4d01f88f8b05");

        // Context classification system
        classification.setSystem(new ClassificationSystemRef(CONTEXT_CLASSIFICATION_SYSTEM_UUID));

        // As the key set is not known get the sealing process to automatically fill this in
        classification.setKeySet(null);

        // URL sealed into content that tells the desktop where to go to get licenses
        classification.setUri(serverURI);

        // Classification time set explicitly to the current time
        classification.setClassificationTime(new Date());

        // As the labels are not known get the sealing process to automatically fill these in
        classification.setLabels(null);

        // Set the context and item code details
        classification.setCookie(cookie);

        // The classification is the only mandatory property for sealing options
        SealingOptions sealingOptions = new SealingOptions();

        sealingOptions.setClassification(classification);

        // Get the MIME type of the file to seal, this is inferred from the unsealed file name
        ContentType contentType = getContentTypeFromPath(unsealedFilename);

        String mimeType = contentType.getMimeTypes()[0];

        // Seal the file
        FileInputStream unsealedInputStream = new FileInputStream(unsealedFilename);

        // Get the sealing services web service
        SealingServicesEndpoint sealingServices = getSealingServicesEndpoint(hostPort, username,
password);

        InputStream sealedInputStream = sealingServices.seal(unsealedInputStream, mimeType,
sealingOptions);

        // Close the file stream
        unsealedInputStream.close();

        // Get the sealed equivalent of the unsealed filename
```

```
        String sealedFilename = getSealedFileName(unsealedFilename);

        // Write the sealed stream out to a file
        FileOutputStream sealedOutputStream = new FileOutputStream(sealedFilename);

        int start = 0;
        int read = 0;

        byte buffer[] = new byte[8194];

        while ((read = sealedInputStream.read(buffer, start, buffer.length)) != -1) {
            sealedOutputStream.write(buffer, 0, read);
        }

        sealedInputStream.close();
        sealedOutputStream.close();
    }
}
```

6.3.12 Peeking a Sealed File

The following code demonstrates how to extract the metadata from sealed content using the peek method. This method sends the sealed content to the sealing server, the server extracts the metadata and returns this information to the caller. The sealed content can be provided as any type of `InputStream`; this example uses a file input stream. Once peeked the file metadata, which includes the Classification details, can be examined. The sample code prints out the human readable classification details (the labels) that were sealed into the content.

Example 6–28

```
import static
oracle.irm.j2ee.jws.content.sealing.SealingServices.getSealingServicesEndpoint;

 import java.io.FileInputStream;

 import oracle.irm.engine.types.content.sealing.ContentDescription;
 import oracle.irm.engine.types.core.classification.Classification;
 import oracle.irm.engine.types.core.general.Label;
 import oracle.irm.j2ee.jws.content.sealing.SealingServicesEndpoint;

 public class PeekFile {

     public static void main(String[] args) throws Exception {

         String hostPort = args[0];
         String username = args[1];
         String password = args[2];

         // The name of the file to peek
         String unsealedFilename = args[3];

         // Get the sealing services web service
         SealingServicesEndpoint sealingServices =
getSealingServicesEndpoint(hostPort, username, password);

         // Perform the peek, providing a stream to the sealed file
         FileInputStream stream = new FileInputStream(unsealedFilename);

         ContentDescription contentDescription = sealingServices.peek(stream);
```

```
        // Close the file stream
        stream.close();

        // Extract the classification details from the content
        Classification classification = contentDescription.getClassification();

        // Show all the labels sealed into content (assumes labels are available)
        for (Label label : classification.getLabels()) {
            System.out.println(label.getLocale().getDisplayName() + " : " +
label.getName());
        }
    }
 }
```

6.3.13 Peeking a Sealed File and Checking the Digital Signature

The following code demonstrates how to extract the metadata from sealed content using the validatedPeek method. This method sends the sealed content to the sealing server, the server extracts the metadata and returns this information to the caller. The sealed content can be provided as any type of `InputStream`; this example uses a file input stream. Once peeked the file metadata, which includes the Classification details, can be examined. The sample code prints out the human readable classification details (the labels) that were sealed into the content.

Example 6–29

```
import static
oracle.irm.j2ee.jws.content.sealing.SealingServices.getSealingServicesEndpoint;

 import java.io.FileInputStream;

 import oracle.irm.engine.types.content.sealing.ContentDescription;
 import oracle.irm.engine.types.core.classification.Classification;
 import oracle.irm.engine.types.core.general.Label;
 import oracle.irm.j2ee.jws.content.sealing.SealingServicesEndpoint;

 public class ValidatedPeekFile {

     public static void main(String[] args) throws Exception {

         String hostPort = args[0];
         String username = args[1];
         String password = args[2];

         // The name of the file to peek
         String unsealedFilename = args[3];

         // Get the sealing services web service
         SealingServicesEndpoint sealingServices =
getSealingServicesEndpoint(hostPort, username, password);

         // Perform the peek, providing a stream to the sealed file
         FileInputStream stream = new FileInputStream(unsealedFilename);

         ContentDescription contentDescription =
sealingServices.validatedPeek(stream);

         // Close the file stream
         stream.close();
```

```
        // Extract the classification details from the content
        Classification classification = contentDescription.getClassification();

        // Show all the labels sealed into content (assumes labels are available)
        for (Label label : classification.getLabels()) {
            System.out.println(label.getLocale().getDisplayName() + " : " +
label.getName());
        }
    }
 }
```

6.3.14 Changing Item Restrictions Associated with a Right

The following code demonstrates how to alter the item locks or exclusions associated with a right. The sample code replaces one item code with two item codes.

Example 6–30

```
import static
oracle.irm.j2ee.jws.rights.context.DocumentRightOperations.getDocumentRightOperati
onsEndpoint;

 import java.net.URLEncoder;
 import java.util.Date;

 import oracle.irm.engine.types.classifications.item.ItemCode;
 import oracle.irm.engine.types.core.account.AccountRef;
 import oracle.irm.engine.types.rights.context.DocumentRight;
 import oracle.irm.engine.types.rights.context.DocumentRightRef;
 import oracle.irm.j2ee.jws.rights.context.DocumentRightOperationsEndpoint;

 public class SaveChangesToItemsWS {

     public static void main(String[] args) throws Exception {

         String hostPort = args[0];
         String username = args[1];
         String password = args[2];

         // Get the document right operations web service
         DocumentRightOperationsEndpoint rightOperations =
getDocumentRightOperationsEndpoint(hostPort, username, password);

         // Reference the account by user name
         AccountRef accountRef = new AccountRef("urn:user:" +
URLEncoder.encode(username, "utf-8"));

         // Get all rights assigned to the account
         DocumentRight[] rights = rightOperations.listRightsByAccount(accountRef);

         // Take the first one on the list
         DocumentRight right = rights[0];

         DocumentRightRef rightRef = new DocumentRightRef(right.getUuid());

         // The save change method allows items to be added and/or removed in the
same call.
         // It does this be comparing two sets of items and applying the
differences.
```

```
        // Item codes
        ItemCode sampleItemCode = new ItemCode();
        sampleItemCode.setValue("sample-item-code");

        ItemCode sampleItemCodeOne = new ItemCode();
        sampleItemCodeOne.setValue("sample-item-code-one");
        sampleItemCodeOne.setTime(new Date());

        ItemCode sampleItemCodeTwo = new ItemCode();
        sampleItemCodeTwo.setValue("sample-item-code-two");
        sampleItemCodeTwo.setTime(new Date());

        // This example shows a delta where item "sample-item-code" is removed
        // and items "sample-item-code-one" and "sample-item-code-two" are added.
        ItemCode[] itemCodes = new ItemCode[] { sampleItemCode };
        ItemCode[] deltaItemCodes = new ItemCode[] { sampleItemCodeOne,
sampleItemCodeTwo };

        // Alter the items
        rightOperations.saveChangesToItems(new DocumentRightRef[] { rightRef
},itemCodes, deltaItemCodes);
    }
 }
```

6.3.15 Unassigning Rights Assigned to a User

The following code demonstrates how to unassign rights that have been assigned to a user. The sample first lists all the rights directly assigned to the user and unassigns them. To unassign the right the authenticated user must be a context manager for the related context.

Example 6–31

```
import static
oracle.irm.j2ee.jws.rights.context.DocumentRightOperations.getDocumentRightOperati
onsEndpoint;

 import java.net.URLEncoder;

 import oracle.irm.engine.types.core.account.AccountRef;
 import oracle.irm.engine.types.rights.context.DocumentRight;
 import oracle.irm.engine.types.rights.context.DocumentRightRef;
 import oracle.irm.j2ee.jws.rights.context.DocumentRightOperationsEndpoint;

 public class UnassignRightsWS {

     public static void main(String[] args) throws Exception {

         String hostPort = args[0];
         String username = args[1];
         String password = args[2];

         // Get the document right operations web service
         DocumentRightOperationsEndpoint rightOperations =
getDocumentRightOperationsEndpoint(hostPort, username, password);

         // Reference the account by user name
         AccountRef accountRef = new AccountRef("urn:user:" +
URLEncoder.encode(username, "utf-8"));
```

```
                    // Get all rights assigned to the account
                    DocumentRight[] rights = rightOperations.listRightsByAccount(accountRef);

                    DocumentRightRef[] rightRefs = new DocumentRightRef[rights.length];

                    for (int i = 0; i < rightRefs.length; ++i) {
                        rightRefs[i] = new DocumentRightRef(rights[i].getUuid());
                    }

                    // Unassign the rights
                    rightOperations.unassignRights(rightRefs);
                }
            }
```

6.3.16 Reclassifying a File

The following code demonstrates how to reclassify a sealed file using the reclassify method. The content to reclassify can be provided as any type of `InputStream`; this example uses a file input stream. The sample changes the labels of the classification and then writes the resulting stream out as a file.

Example 6–32

```
import static oracle.irm.j2ee.jws.content.sealing.SealingServices.getSealingServicesEndpoint;

 import java.io.FileInputStream;
 import java.io.FileOutputStream;
 import java.io.InputStream;
 import java.util.Locale;

 import oracle.irm.engine.types.content.sealing.ContentDescription;
 import oracle.irm.engine.types.core.classification.Classification;
 import oracle.irm.engine.types.core.general.Label;
 import oracle.irm.j2ee.jws.content.sealing.SealingServicesEndpoint;

 public class ReclassifyFile {

        public static void main(String[] args) throws Exception {

         String hostPort = args[0];
         String username = args[1];
         String password = args[2];

         // Get the file to reclassify
         String filename = args[3];

         // Get the label to apply to the classification
         String labelName = args[4];

         // Get a sealing services end point
         SealingServicesEndpoint sealingServices =
getSealingServicesEndpoint(hostPort,username,password);

         // Peek the contents of the file to obtain the classification details of the provided file
         FileInputStream inputStream = new FileInputStream(filename);

         ContentDescription contentDescription = sealingServices.peek(inputStream);

         inputStream.close();
```

```
        // Extract the classification from the content description
        Classification classification = contentDescription.getClassification();

        // Replace the labels with one
        Label label = new Label(
            Locale.ENGLISH,
            labelName,
            null);

        classification.setLabels(new Label[] { label });

        // Reclassify the sealed file with the new classification
        inputStream = new FileInputStream(filename);

        InputStream reclassifiedStream = sealingServices.reclassify(inputStream,classification);

        inputStream.close();

        // Write the stream out to a file
        FileOutputStream reclassifiedOutputStream = new FileOutputStream(filename);

        int start = 0;
        int read = 0;

        byte buffer[] = new byte[8194];

        while ((read = reclassifiedStream.read(buffer, start, buffer.length)) != -1)
        {
            reclassifiedOutputStream.write(buffer, 0, read);
        }

        reclassifiedStream.close();
        reclassifiedOutputStream.close();
       }
}
```

6.3.17 Resealing a File with Different Custom Data

The following code demonstrates how to reseal a sealed file using the reseal method. The content to reseal can be provided as any type of `InputStream`; this example uses a file input stream. The sample adds XML-based custom data to the sealed file.

Example 6–33

```
import static oracle.irm.j2ee.jws.content.sealing.SealingServices.getSealingServicesEndpoint;

 import java.io.FileInputStream;
 import java.io.FileOutputStream;
 import java.io.InputStream;
 import java.util.UUID;

 import javax.xml.parsers.DocumentBuilder;
 import javax.xml.parsers.DocumentBuilderFactory;

 import oracle.irm.engine.types.content.sealing.CustomData;
 import oracle.irm.j2ee.jws.content.sealing.SealingServicesEndpoint;

 import org.w3c.dom.Document;
 import org.w3c.dom.Element;
```

```
 public class ResealFile {

        public static void main(String[] args) throws Exception {

         String hostPort = args[0];
         String username = args[1];
         String password = args[2];

         // Get the file to reseal
         String filename = args[3];

         // Get a sealing services end point
         SealingServicesEndpoint sealingServices =
getSealingServicesEndpoint(hostPort,username,password);

         // Custom data is provided as XML
         DocumentBuilderFactory documentBuilderFactory = DocumentBuilderFactory.newInstance();
         DocumentBuilder documentBuilder = documentBuilderFactory.newDocumentBuilder();
         Document document = documentBuilder.newDocument();

         Element element = document.createElement("SampleCustomData");
         element.setTextContent("Some example custom data provided as an XML text element");

         CustomData data = new CustomData();

         // UUID identifies the custom data, in this case just use a random UUID value
         data.setUuid(UUID.randomUUID());

         // Custom data is XML document
         data.setData(element);

         // Reseal the sealed file with the new custom data
         InputStream inputStream = new FileInputStream(filename);

         InputStream reclassifiedStream = sealingServices.reseal(inputStream, new CustomData[]
{data});

         inputStream.close();

         // Write the stream out to a file
         FileOutputStream reclassifiedOutputStream = new FileOutputStream(filename);

         int start = 0;
         int read = 0;

         byte buffer[] = new byte[8194];

         while ((read = reclassifiedStream.read(buffer, start, buffer.length)) != -1)
         {
             reclassifiedOutputStream.write(buffer, 0, read);
         }

         reclassifiedStream.close();
         reclassifiedOutputStream.close();
        }
 }
```

6.3.18 Unsealing a File

The following code demonstrates how to unseal a sealed file using the unseal method. The content to unseal can be provided as any type of `InputStream`; this example uses a file input stream. The sample writes the resulting unsealed stream out to a file.

Example 6–34

```
import static
oracle.irm.j2ee.jws.content.sealing.SealingServices.getSealingServicesEndpoint;

 import java.io.FileInputStream;
 import java.io.FileOutputStream;
 import java.io.InputStream;

 import oracle.irm.j2ee.jws.content.sealing.SealingServicesEndpoint;

 public class UnsealFile {

     public static void main(String[] args) throws Exception {

         String hostPort = args[0];
         String username = args[1];
         String password = args[2];

         // The file to unseal
         String sealedFilename = args[3];

         // The unsealed file name
         String unsealedFilename = args[4];

         // Get the sealing services web service
         SealingServicesEndpoint sealingServices =
getSealingServicesEndpoint(hostPort, username, password);

         // Unseal the sealed file
         FileInputStream sealedFileStream = new FileInputStream(sealedFilename);

         InputStream unsealedStream = sealingServices.unseal(sealedFileStream);

         // Close the file stream
         sealedFileStream.close();

         // Write the stream out to a file
         FileOutputStream unsealedStreamOutputStream = new
FileOutputStream(unsealedFilename);

         int start = 0;
         int read = 0;

         byte buffer[] = new byte[8194];

         while ((read = unsealedStream.read(buffer, start, buffer.length)) != -1)
{
             unsealedStreamOutputStream.write(buffer, 0, read);
         }

         // Close the streams
         unsealedStream.close();
         unsealedStreamOutputStream.close();
```

```
    }
}
```

6.3.19 Listing Classifications

The following code demonstrates how to list classification details from a sealing server using the listClassifications method. The sample code displays the list of Classifications details available to the authenticated user.

Example 6–35

```
import static
oracle.irm.j2ee.jws.core.storage.DesktopServices.getDesktopServicesEndpoint;

 import java.net.URI;

 import oracle.irm.engine.types.core.classification.Classification;
 import oracle.irm.engine.types.core.general.Label;
 import oracle.irm.j2ee.jws.core.storage.DesktopServicesEndpoint;

 public class ListClassificationsWS {

     public static void main(String[] args) throws Exception {

         String hostPort = args[0];
         String username = args[1];
         String password = args[2];

         // The server URI. for example https://irm.example.com/irm_desktop
         URI serverURI = URI.create(args[3]);

         // Get the desktop services web service
         DesktopServicesEndpoint desktopServices =
getDesktopServicesEndpoint(hostPort, username, password);

         // Synchronize with the specified server
         Classification[] classifications =
desktopServices.listClassifications(serverURI);

         // Display the labels of the classifications
         for (Classification classification : classifications) {

             for (Label label : classification.getLabels()) {
                 System.out.println(label.getLocale().getDisplayName() + " : " +
label.getName());
             }
         }
     }
 }
```

6.3.20 Searching the Context Journal Using Web Services

The following code demonstrates how to search the content usage for context classified content. The sample code searches for all entries for the last twenty-four hours and displays a short summary for the first one hundred entries.

Example 6–36

```
import static
oracle.irm.j2ee.jws.rights.context.ContextOperations.getContextOperationsEndpoint;
```

```
import java.util.Calendar;
import java.util.Date;

import oracle.irm.engine.types.core.general.PageRange;
import oracle.irm.engine.types.core.time.TimeRange;
import oracle.irm.engine.types.rights.journal.ContextJournalEntry;
import oracle.irm.j2ee.jws.rights.context.ContextOperationsEndpoint;

public class SearchJournalWS {

    public static void main(String[] args) throws Exception {

        // The server address. for example https://localhost
        String hostPort = args[0];
        String username = args[1];
        String password = args[2];

        // Search for all records from the last 24 hours
        Date end = new Date();

        // Use a calendar to work out the time range
        Calendar calendar = Calendar.getInstance();

        calendar.setTime(end);
        calendar.add(Calendar.DAY_OF_MONTH, -1);

        Date begin = calendar.getTime();

        TimeRange timeRange = new TimeRange(begin, end);

        // Get the context operations web service
        ContextOperationsEndpoint contextOperations =
getContextOperationsEndpoint(hostPort, username, password);

        // Search the context journal
        PageRange pageRange = new PageRange(1, 100);

        ContextJournalEntry[] journalResults = contextOperations.searchJournal(
            null, // no accounts filter
            null, // no item codes filter
            timeRange,
            pageRange,
            null); // no sorting details

        if (journalResults.length == 0)
            return;

        // Display the timestamp, URI and and feature for each entry
        for (ContextJournalEntry entry : journalResults) {

            System.out.print("Timestamp : " + entry.getTime());
            System.out.print("Account   : " + entry.getAccount().getName());
            System.out.print("Content   : " + entry.getUri() != null ?
entry.getUri() : "");
            System.out.print("Feature   : " + entry.getFeature());
        }
    }
}
```

6.3.21 Checking in Licenses

The following code demonstrates how to check in licenses currently checked out to a sealing server. When the sealing server processes content it will usually check out licenses for the authenticated user. These licenses can no longer be used from other locations (for example, the Oracle IRM Desktop) until they expire or are manually checked in.

Example 6–37

```
import static
oracle.irm.j2ee.jws.core.storage.DesktopServices.getDesktopServicesEndpoint;

 import java.net.URI;

 import oracle.irm.j2ee.jws.core.storage.DesktopServicesEndpoint;

 public class CheckInWS {

     public static void main(String[] args) {

         String hostPort = args[0];
         String username = args[1];
         String password = args[2];

         // Get the desktop services web service
         DesktopServicesEndpoint desktopServices =
getDesktopServicesEndpoint(hostPort, username, password);

         // The server URI. for example https://irm.example.com/irm_desktop
         URI serverURI = URI.create(args[3]);

         // Check in all the licenses currently within the
         // desktop store for the given server
         desktopServices.checkIn(serverURI);
     }
```

6.3.22 Deleting a Domain

The following code demonstrates how to delete a domain. The sample code uses a fixed domain UUID for the new domain so that all sample code can work with a known domain. A new domain would typically be given a new random UUID value. The authenticated user must be a domain administrator. When a domain is deleted all the associated roles, templates and contexts are also deleted.

Example 6–38

```
import static
oracle.irm.j2ee.jws.rights.context.DomainOperations.getDomainOperationsEndpoint;

 import java.util.UUID;

 import oracle.irm.engine.types.rights.context.DomainRef;
 import oracle.irm.j2ee.jws.rights.context.DomainOperationsEndpoint;

 public class DeleteDomainWS {

     public static void main(String[] args) throws Exception {

         String hostPort = args[0];
```

```
        String username = args[1];
        String password = args[2];

        // Domain UUID is fixed for sample code
        UUID domainUUID =
UUID.fromString("6fab93fd-2858-461a-a0b3-34e261dbf8fd");

        // Get the domain operations web service
        DomainOperationsEndpoint domainOperations =
getDomainOperationsEndpoint(hostPort, username, password);

        DomainRef domain = new DomainRef(domainUUID);

        // Delete the domain using the domain reference
        domainOperations.deleteDomain(domain);
    }
 }
```

7

Code Samples for Java Applications

This section contains the following:

7.1 Introduction

Required Jar Files

These code samples require the following jar file to be added to the class path:

- irm-api.jar

Local Sealing Operations

Certain operations on sealed content can be performed locally within a J2SE or J2EE application. These local versions are provided as an alternative to sending the content to a server to be processed.

Currently the following operations are supported:

7.2 Sealing a File

The following code demonstrates how to seal a file using the `seal` method. The content to seal can be provided as any type of `java.io.InputStream`; this example uses a file input stream. Similarly, the sealed content can be written out to any output

stream; this example writes the sealed content as a file whose file name is derived from the unsealed file name. When a file is sealed, a `Classification` must be specified. In this sample the file is sealed using the context classification system, specifying a context with a known UUID value and a fixed item code value.

Example 7–1

```
import static oracle.irm.engine.classifications.context.ContextConstants.CONTEXT_
CLASSIFICATION_SYSTEM;
 import static
oracle.irm.engine.classifications.context.ContextCookieFactory.createContextCookie
;
 import static
oracle.irm.engine.classifications.context.ContextFactory.createContext;
 import static
oracle.irm.engine.classifications.item.ItemCodeFactory.createItemCode;
 import static oracle.irm.engine.content.sealing.SealingOperationsInstance.seal;
 import static
oracle.irm.engine.content.sealing.SealingOptionsFactory.createSealingOptions;
 import static
oracle.irm.engine.content.source.FileSourceFactory.createFileSource;
 import static
oracle.irm.engine.content.type.ContentTypeOperationsInstance.getSealedFileName;
 import static
oracle.irm.engine.core.classification.ClassificationFactory.createClassification;
 import static oracle.irm.engine.core.general.LabelCollectionFactory.EMPTY_LABELS;

 import java.io.FileOutputStream;
 import java.net.Authenticator;
 import java.net.PasswordAuthentication;
 import java.net.URI;
 import java.util.Date;
 import java.util.UUID;

 import oracle.irm.engine.classifications.context.Context;
 import oracle.irm.engine.classifications.context.ContextCookie;
 import oracle.irm.engine.classifications.item.ItemCode;
 import oracle.irm.engine.content.sealing.SealingOptions;
 import oracle.irm.engine.content.source.FileSource;
 import oracle.irm.engine.core.classification.Classification;

 public class SealFile {

     public static void main(String[] args) throws Exception {

         // The user name and password are provided on the command line. In a
production
         // system these details should be provided in a more secure manner, such
         // as prompting from the console, or reading from a secure source.
         final String username = args[0];
         final String password = args[1];

         // Configure an authenticator to provide the credentials for any network
access
         Authenticator.setDefault(new Authenticator() {
             @Override
             protected PasswordAuthentication getPasswordAuthentication() {
                 return new PasswordAuthentication(username,
password.toCharArray());
             }
```

```
        });

        // Provide an explicit item code for the document
        ItemCode itemCode = createItemCode("sample document");

        // Context UUID is fixed for sample code
        Context context =
createContext(UUID.fromString("46f910d9-dd30-476e-b060-4d01f88f8b05"));

        // Context cookie specifying the context and the item code
        ContextCookie cookie = createContextCookie(
            context,
            itemCode);

        // The server address e.g. https://irm.example.com/irm_desktop
        URI serverURI = URI.create(args[2]);

        // Create the classification details used in the sealing options
        Classification classification = createClassification(
            "46f910d9-dd30-476e-b060-4d01f88f8b05",
            CONTEXT_CLASSIFICATION_SYSTEM,
            null, // automatically fill in key set
            serverURI,
            new Date(),
            EMPTY_LABELS, // automatically fill in labels
            cookie);

        // Create the sealing options
        SealingOptions sealingOptions = createSealingOptions(classification);

        // Create a file source from the file name
        String unsealedFilename = args[3];

        FileSource fileSource = createFileSource(unsealedFilename);

        // Get the sealed equivalent of the unsealed filename
        String sealedFilename = getSealedFileName(unsealedFilename);

        // Write the sealed stream out to a file
        FileOutputStream sealedOutputStream = new
FileOutputStream(sealedFilename);

        // Seal the file
        seal(fileSource, sealedOutputStream, sealingOptions);

        // Close the streams
        sealedOutputStream.close();
    }
 }
```

7.3 Peeking a Sealed File

The following code demonstrates how to extract the metadata from sealed content using the `peek` method. The sealed content can be provided as any type of `java.io.InputStream`; this example uses a file input stream. Once peeked, the file metadata, which includes the `Classification` details, can be examined. The sample code prints out the human-readable classification details (the labels) that were sealed into the content.

Th `peek` method does not attempt to check the public header against its declared signature. If the metadata has been altered post-sealing, this method will not throw an exception.

Example 7–2

```
import static oracle.irm.engine.content.sealing.SealingOperationsInstance.peek;

 import java.io.FileInputStream;
 import java.io.IOException;

 import oracle.irm.engine.content.sealing.ContentDescription;
 import oracle.irm.engine.core.classification.Classification;
 import oracle.irm.engine.core.general.Label;

 public class PeekFile {

     public static void main(String[] args) throws IOException {

         // The name of the file to peek is the first
         // command line argument
         FileInputStream stream = new FileInputStream(args[0]);

         // Perform the peek, providing a stream to the sealed file
         ContentDescription contentDescription = peek(stream);

         // Close the file stream
         stream.close();

         // Extract the classification details from the content
         Classification classification = contentDescription.getClassification();

         // Show all the labels sealed into content
         for (Label label : classification.getLabels()) {
             System.out.println(label.getLocale().getDisplayName() + " : " +
label.getName());
         }
     }
 }
```

7.4 Peeking a Sealed File and Checking the Digital Signature

The following code demonstrates how to extract the metadata from sealed content using the `validatedPeek` method. Once peeked, the file metadata, which includes the `Classification` details, can be examined. The sample code prints out the human readable classification details (the labels) that were sealed into the content.

The `validatedPeek` method attempts to check the public header against its declared signature. If the public header metadata has been altered post-sealing, this method will throw an exception.

Example 7–3

```
import static
oracle.irm.engine.content.sealing.SealingOperationsInstance.validatedPeek;

 import java.io.FileInputStream;
 import java.io.IOException;
 import java.net.Authenticator;
 import java.net.PasswordAuthentication;
```

```
 import oracle.irm.engine.content.sealing.ContentDescription;
 import oracle.irm.engine.core.classification.Classification;
 import oracle.irm.engine.core.general.Label;

 public class ValidatedPeekFile {

     public static void main(String[] args) throws IOException {

         // The user name and password are provided on the command line. In a
production
         // system these details should be provided in a more secure manner, such
         // as prompting from the console, or reading from a secure source.
         final String username = args[0];
         final String password = args[1];

         // Configure an authenticator to provide the credentials for any network
access
         Authenticator.setDefault(new Authenticator() {
             @Override
             protected PasswordAuthentication getPasswordAuthentication() {
                 return new PasswordAuthentication(username,
password.toCharArray());
             }
         });

        // The name of the file to peek
         FileInputStream stream = new FileInputStream(args[2]);

         // Perform the peek, providing a stream to the sealed file
         ContentDescription contentDescription = validatedPeek(stream);

         // Close the file stream
         stream.close();

         // Extract the classification details from the content
         Classification classification = contentDescription.getClassification();

         // Show all the labels sealed into content
         for (Label label : classification.getLabels()) {
             System.out.println(label.getLocale().getDisplayName() + " : " +
label.getName());
         }
     }
 }
```

7.5 Reclassifying a File

The following code demonstrates how to reclassify a sealed file using the `reclassify` method. The content to reclassify can be provided as any type of `java.io.InputStream`; this example uses a file input stream. The sample code changes the labels of the classification and then writes the resulting reclassified stream out as a file.

Example 7–4

```
import static oracle.irm.engine.content.sealing.SealingOperationsInstance.peek;
 import static
oracle.irm.engine.content.sealing.SealingOperationsInstance.reclassify;
```

```
 import static
oracle.irm.engine.core.classification.ClassificationFactory.createClassification;
 import static oracle.irm.engine.core.general.LabelFactory.createLabel;

 import java.io.FileInputStream;
 import java.io.FileOutputStream;
 import java.net.Authenticator;
 import java.net.PasswordAuthentication;
 import java.util.Collections;
 import java.util.Locale;

 import oracle.irm.engine.content.sealing.ContentDescription;
 import oracle.irm.engine.core.classification.Classification;
 import oracle.irm.engine.core.general.Label;

 public class ReclassifyFile {

     public static void main(String[] args) throws Exception {

         // The user name and password are provided on the command line. In a
production
         // system these details should be provided in a more secure manner, such
         // as prompting from the console, or reading from a secure source.
         final String username = args[0];
         final String password = args[1];

         // Configure an authenticator to provide the credentials for any network
access
         Authenticator.setDefault(new Authenticator() {
             @Override
             protected PasswordAuthentication getPasswordAuthentication() {
                 return new PasswordAuthentication(username,
password.toCharArray());
             }
         });

         // The file to reclassify
         String filename = args[2];

         // The output file name
         String reclassifiedFilename = args[3];

         // The label to apply to the classification
         String labelName = args[4];

         // Peek the contents of the file to obtain the current classification
         FileInputStream originalFileStream = new FileInputStream(filename);

         ContentDescription contentDescription = peek(originalFileStream);

         // Close the file stream
         originalFileStream.close();

         // Extract the classification from the content description
         Classification classification = contentDescription.getClassification();

         // Replace the labels with the one specified
         Label label = createLabel(Locale.ENGLISH, labelName, null);

         classification = createClassification(
```

```
                classification.getId(),
            classification.getSystem(),
            classification.getKeySet(),
            classification.getUri(),
            classification.getClassificationTime(),
            Collections.singleton(label),
            classification.getCookie());

        // Reclassify the sealed file with the new classification
        originalFileStream = new FileInputStream(filename);

        FileOutputStream reclassifiedFileStream = new
FileOutputStream(reclassifiedFilename);

        reclassify(originalFileStream, reclassifiedFileStream, classification);

        // Close the streams
        originalFileStream.close();
        reclassifiedFileStream.close();
    }
 }
```

7.6 Resealing a File with Different Custom Data

The following code demonstrates how to reseal a sealed file using the `reseal` method. This sample adds XML-based custom data to the sealed file.

Example 7–5

```
 import static
oracle.irm.engine.content.sealing.CustomDataCollectionFactory.createCustomData;
 import static
oracle.irm.engine.content.sealing.CustomDataFactory.createCustomData;
 import static oracle.irm.engine.content.sealing.SealingOperationsInstance.reseal;

 import java.io.FileInputStream;
 import java.io.FileOutputStream;
 import java.net.Authenticator;
 import java.net.PasswordAuthentication;
 import java.util.Collection;
 import java.util.UUID;

 import javax.xml.parsers.DocumentBuilder;
 import javax.xml.parsers.DocumentBuilderFactory;

 import oracle.irm.engine.content.sealing.CustomData;

 import org.w3c.dom.Document;
 import org.w3c.dom.Element;

 public class ResealFile {

     public static void main(String[] args) throws Exception {

         // The user name and password are provided on the command line. In a
production
         // system these details should be provided in a more secure manner, such
         // as prompting from the console, or reading from a secure source.
         final String username = args[0];
         final String password = args[1];
```

```
        // Configure an authenticator to provide the credentials for any network
access
        Authenticator.setDefault(new Authenticator() {
            @Override
            protected PasswordAuthentication getPasswordAuthentication() {
                return new PasswordAuthentication(username,
password.toCharArray());
            }
        });

        // The file to reseal
        String filename = args[2];

        // The output file name
        String resealedFilename = args[3];

        // Custom data is provided as XML
        DocumentBuilderFactory documentBuilderFactory =
DocumentBuilderFactory.newInstance();
        DocumentBuilder documentBuilder =
documentBuilderFactory.newDocumentBuilder();
        Document document = documentBuilder.newDocument();

        Element element = document.createElement("SampleCustomData");
        element.setTextContent("Some example custom data provided as an XML text
element");

        // UUID identifies the custom data, in this example just use a random
UUID value
        UUID uuid = UUID.randomUUID();

        // Create the custom data, UUID + value
        CustomData data = createCustomData(uuid, element);

        Collection customData = createCustomData(data);

        // Reclassify the sealed file with the new classification
        FileInputStream sealedFileStream = new FileInputStream(filename);

        FileOutputStream resealedFileStream = new
FileOutputStream(resealedFilename);

        reseal(sealedFileStream, resealedFileStream, customData);

        // Close the streams
        sealedFileStream.close();
        resealedFileStream.close();
    }
 }
```

7.7 Unsealing a File

The following code demonstrates how to unseal a sealed file using the `unseal` method. The content to unseal can be provided as any type of `java.io.InputStream`; this example uses a file input stream. The sample code writes the resulting stream out to a file.

Example 7–6

```
import static oracle.irm.engine.content.sealing.SealingOperationsInstance.unseal;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.net.Authenticator;
import java.net.PasswordAuthentication;

public class UnsealFile {

    public static void main(String[] args) throws Exception {

        // The user name and password are provided on the command line. In a
production
        // system these details should be provided in a more secure manner, such
        // as prompting from the console, or reading from a secure source.
        final String username = args[0];
        final String password = args[1];

        // Configure an authenticator to provide the credentials for any network
access
        Authenticator.setDefault(new Authenticator() {
            @Override
            protected PasswordAuthentication getPasswordAuthentication() {
                return new PasswordAuthentication(username,
password.toCharArray());
            }
        });

        // The file to unseal
        String sealedFilename = args[2];

        // The unsealed file name
        String unsealedFilename = args[3];

        // Sealed file input stream
        FileInputStream inputStream = new FileInputStream(sealedFilename);

        // Unsealed file output stream
        FileOutputStream outputStream = new FileOutputStream(unsealedFilename);

        // Unseal the sealed file
        unseal(inputStream, outputStream);

        // Close the file streams
        inputStream.close();
        outputStream.close();
    }
}
```

8

Status Page Customization

This section contains the following topics:

8.1 Overview

Oracle IRM displays status pages to users whenever they are denied access to sealed content. This may be because they do not have the correct rights to view the content, or because their rights have expired and could not be refreshed.

Status pages are HTML pages displayed by Oracle IRM Desktop (the Oracle IRM client application) in a dialog containing an embedded instance of Microsoft Internet Explorer. There are offline and online status pages. Offline pages are generated locally by Oracle IRM Desktop, are of a standard layout, and contain only information that Oracle IRM Desktop has available. Online pages are generated by a web server, where Oracle IRM Server resides.

If the Oracle IRM Desktop computer is not connected to the network, then the offline page is displayed. If the computer is connected to the network, then Oracle IRM Desktop will try to display an online page, and if that fails to load it will fall back to displaying an offline page.

Note: Only online status pages are customizable.

When Oracle IRM Desktop wants to show an online status page, it sends an XML document (via a POST) to a status page hosted on the web server where Oracle IRM Server resides. The standard response is to render an appropriate status page using the XML data. For example, if the user has no licenses, the status page will show a message such as "No licenses available for Top Secret context", using the XML to determine what content was being opened. For customers that wish to provide their own status pages, configuration settings are provided on the Oracle IRM pages of the Oracle Enterprise Manager Fusion Middleware Control Console.

8.2 Customizing Status Pages

Customizing status pages is achieved by hosting equivalents of the standard status pages on your own web server. This gives full flexiblility of what can be displayed on a custom status page.

When you have created custom status pages, Oracle IRM Server can be made to redirect status page requests to a specified URL. The redirection includes information about the operation that was being performed by Oracle IRM Desktop. This allows the custom status pages to be at least as informative as the supplied status pages.

The redirection of the status page requests can be performed using either of two HTTP methods: HTTP GET or HTTP POST.

8.2.1 Redirection of Status Page Requests Using HTTP GET

When the redirection of status page requests is performed using HTTP GET, all of the information about the state of Oracle IRM Desktop must be encoded in the query string of the URL that is to serve the custom page. To avoid overly long query strings containing information that is not required by the custom page, only the configured redirection URL acts as a template for the URL that is used.

For example, consider a status page that is only concerned with the type of status page requested and the classification name. If the page is to be served by a page at `http://some.example.com/statusPage` then the URL could be configured as below:

```
http://some.example.com/statusPage?page=&irm-classification-name=
```

When the redirection is for a classification called 'Top Secret' and where a license has expired, the request would be populated as below:

```
http://some.example.com/statusPage?page=LICENSE_
EXPIRED=&irm-classification-name=Top+Secret
```

The parameters can be chosen from a list of built-in parameters, and from all of the content attributes supported by the server. In the above example, `page` is a built-in parameter and `irm-classification-name` is a content attribute.

Note: You can design the redirection query string to contain other parameters. Parameters that have already been populated by Oracle IRM and parameters not recognized by Oracle IRM will be left untouched.

The maximum size of URL that can be processed by Microsoft Internet Explorer is 2048 characters.

8.2.2 Redirection of Status Page Requests Using HTTP POST

The HTTP POST method sends all of the available Oracle IRM Desktop state information as a form to the configured URL. The advantage is that all of the Oracle IRM Desktop state information is available without having to consider the size of the URL. JavaScript must be available on the client computer.

The page parameter is passed as part of the URL. So:

```
http://some.example.com/status
```

will result in a post to:

```
http://some.example.com/status?page=LICENSE_EXPIRED
```

The Oracle IRM Desktop state is provided as XML, with the relevant information picked out by the developer of the custom status page. The `page` parameter is still sent as part of the URL query string, because this is how it is received from Oracle IRM Desktop.

See Example 8–1, "Desktop State in XML Form".

8.3 Configuring Oracle IRM for Custom Status Pages

A server-wide setting affects all status page requests relating to the Context classification system. Custom classification systems (if any), and the built-in Oracle IRM test content page system, are not affected by the setting.

Configuration consists of choosing whether to redirect using HTTP GET or HTTP POST, and setting the URL of the server hosting the custom status pages.

To configure using the Oracle IRM pages on Oracle Enterprise Manager Fusion Middleware Control Console, see the *Oracle Fusion Middleware Administrator's Guide for Oracle IRM Server*.

To configure using WLST, see the *Oracle Fusion Middleware WLST Command Reference Guide*.

8.4 Creating Custom Status Pages Using the HTTP GET Method

The developer will need to pick which parameters he will need to build an appropriate status page for the user. The page will then be developed and a URI with the required parameters configured on the Oracle IRM server.

The parameters that are available are all content attributes and the built-in parameters listed in the `StatusPageOperations` Javadoc. The Javadoc for the `populateRedirectionURI` method of `StatusPageOperations` contains details on how the URI is populated to perform the redirect.

8.5 Creating Custom Status Pages Using the HTTP POST Method

The page developer will post a form with an input field called `desktop.state`. This `DesktopState` will be the XML string that Oracle IRM Desktop posted to the Oracle IRM server.

The XML document can be parsed by the page and an appropriate page built. The Javadoc for desktop state contains details on what is in the XML document.

8.6 Reference Information and Examples

This section contains the following:

- List of Built-in Parameters
- List of Status Page Types
- List of URL Elements Available for Use in Custom Status Pages
- Example of Oracle IRM Desktop State in XML

8.6.1 List of Built-in Parameters

Table 8–1 and Table 8–2 contain the built-in parameters that can be used with the HTTP GET method of redirection.

Table 8–1 Built-in Parameters (A)

Parameter	Description
application.container	The container application can be considered the application hosting the desktop logic, such as a browser or word processor.
application.name	The application name can be used to identify the application used to access the sealed content.
desktop.operating.system	The value can be used to determine the operating system that is hosting the desktop logic.
desktop.uuid	The desktop UUID is used to identify a particular desktop independently of the product version number.
desktop.version	The desktop version identifies the product version of the desktop used to access the sealed content.
page	The status pages provided by the desktop web site. See Table 8–3, " Status Page Types".

Table 8–2 Built-in Parameters (B)

Parameter	Name	Description
irm-time	Current Time	The time on the Oracle IRM server.
irm-locale	Desktop Locale	The locale of the desktop installation.
irm-location	Content Location	The location the sealed content was opened on the desktop.
irm-mime	Sealed MIME Type	The sealed content MIME type.
irm-extension	Sealed File Extension	The sealed content file extension.
irm-account-uuid	Account UUID	The UUID value that identifies the user.
irm-account-name	User Name	The user name of the user.
irm-creation-time	Creation Time	The sealed content creation time.
irm-edit-time	Edit Time	The sealed content last edit time.
irm-schema-version	Schema Version	The sealed content schema version.
irm-classification-name	Sealed To	The classification label.
irm-classification-description	Description	The classification label description.
irm-classification-keyset	Key Set UUID	The classification key set UUID that identifies which key set was used to seal the content.
irm-classification-system	Classification System	The classification system UUID. This value identifies the type of classification, for example, context classified content.
irm-classification-time	Classification Time	The classification time is set when sealed content is sealed.
irm-classification-url	Server Address	The server URI sealed into content.

Table 8–2 (Cont.) Built-in Parameters (B)

Parameter	Name	Description
irm-host	Desktop Host Name	The desktop host name.
irm-context-uuid	Context UUID	The UUID value that identifies the context.
irm-context-itemcode-value	Item Code Value	The item code value.
irm-context-itemcode-time	Item Code Time Stamp	The item code time stamp, if provided.

8.6.2 List of Status Page Types

Table 8–3 contains the status page types for use with the **page** built-in parameters in Table 8–1 and Table 8–2.

When Oracle IRM Desktop requests a status page it will set the appropriate status using a query parameter. For example, if the user is using content, but the license they are using expired, Oracle IRM Desktop will request a status page with a query parameter set as `page=LICENSE_EXPIRED`. Table 8–3 lists all the status pages the Oracle IRM desktop can send.

Table 8–3 Status Page Types

Status	Description
DIAGNOSTICS	When the self-test action is performed within the desktop, one of the steps is to contact the desktop web site. In this scenario the desktop will ask for the diagnostic status page.
INFORMATION	When the user clicks on the 'information' button or link the information status page is requested. The information status page should provide details about the content's classification.
GENERAL_ERROR	A general desktop error has occurred.
UNKNOWN	Unknown status page.
PRIVACY	Privacy statement status page.
LICENSE_EXPIRED	When a user is using sealed content, their license-based rights may expired. If the license cannot be refreshed from the server this status page will be displayed.
LICENSES_CHECKED_IN	A license is applicable, but in use on another device. This status page will display details about the other device or devices.
LICENSE_CHANGED	When a user is using sealed content, their license-based rights may be refreshed from the server. If these rights change, for example allowing printing, this status page will be displayed.
NO_LICENSES	The user has no rights to access the content.
NO_LICENSES_OFFLINE	The user has no rights stored off-line to access the content. The server cannot be contacted to see if there are licenses available.
UPGRADE	The server has prompted the desktop to perform a mandatory upgrade.
REPUDIATED	The server has denied access to the desktop.
SERVER_CONNECTION	The server cannot be contacted.
AUTHENTICATION_ ANONYMOUS	The user has accessed content but chosen to cancel the authentication process.

Table 8–3 (Cont.) Status Page Types

Status	Description
AUTHENTICATION_ FAILED	The user has accessed content, authenticated, but failed to authenticate (for example, a bad password).
UNKNOWN_ CLASSIFICATION	The server does not know about the classification of the content. This would typically occur if the classification has been removed from the server after creating sealed content.
UNSUPPORTED_FORMAT	The desktop cannot render the content format. For example, occurs when the application that normally renders the content has not been installed.
OFFICE_PASSWORD_ PROTECTED	The desktop attempts to protect the content with password protection. If this password protection cannot be applied, this status page is displayed.
OFFICE_PLUGIN_NOT_ TRUSTED	A third party plug-in is not trusted and is preventing the sealed content from being accessed.
MOVIE_BEFORE_MOVIE	This status page is displayed before a sealed movie has been started.
MOVIE_AFTER_MOVIE	This status page is displayed after a sealed movie has been shown.

8.6.3 List of URL Elements Available for Use in Custom Status Pages

The following special sinfo: URL elements are available for use in links in your custom status pages, for example `<a href="sinfo:panel">Control Panel</a>`, which would open the Oracle IRM Desktop Control Panel dialog.

Table 8–4 sinfo URL Elements

URL element	Description
sinfo:reason	Displays a message box containing further information. Only relevant to a General Error or Rights in Use status page. ■ General Error: support details for the error. ■ Rights in Use: message box listing the device(s) that the rights are checked out on.
sinfo:reload	Causes Oracle IRM Desktop to open the sealed file again.
sinfo:panel	Opens the Oracle IRM Desktop Control Panel dialog.
sinfo:test	Opens the Oracle IRM Server Connection Test dialog.
sinfo:help	Opens the Oracle IRM Desktop help.
sinfo:systeminfo	Opens the Oracle IRM Desktop Support Information dialog.
sinfo:about	Opens the About dialog.

8.6.4 Example of Oracle IRM Desktop State in XML

The following XML document shows an example Oracle IRM Desktop state in XML form:

Example 8–1 Desktop State in XML Form

```
<?xml version="1.0" encoding="UTF-8"?>
<core:DesktopState xmlns:core="http://xmlns.oracle.com/irm/core">
    <desktop>
        <uuid>70678535-0a6f-4cf9-9411-2c05ed8d989</uuid>
```

```
            <version>
                <version>11.1.1.1.0</version>
            </version>
            <operatingSystem>Microsoft XP SP 2</operatingSystem>
            <locale>en</locale>
            <device>
                <uuid>a7352732-dcd0-43af-93c5-0cbc7c1f203d</uuid>
                <name>machine</name>
            </device>
            <application>
                <name>desktop</name>
                <container>browser</container>
            </application>
        </desktop>
        <contentDescription>
            <schema>
                <schemaVersion>
                    <version>6.0</version>
                </schemaVersion>
            </schema>
            <classification>
                <id>7ec1c191-0531-4876-813e-c554676df09b</id>
                <system>
                    <uuid>588403f9-9cff-4cce-88e4-e030cc57282a</uuid>
                </system>
                <keySet>
                    <uuid>213f8f65-c5d1-4868-9fff-ad156daa2dd6</uuid>
                </keySet>
                <uri>http://irm.example.com/irm_desktop</uri>
                <classifications:ContextCookie
    xmlns:classifications="http://xmlns.oracle.com/irm/classifications">
                    <context>
                        <uuid>588403f9-9cff-4cce-88e4-e030cc57282a</uuid>
                    </context>
                    <itemCode>
                        <value>sample.sdoc</value>
                    </itemCode>
                </classifications:ContextCookie>
                <classificationTime>2008-02-01T13:00:00.000+01:00</classificationTime>
                <labels>
                    <locale>en</locale>
                    <name>Top Secret</name>
                </labels>
            </classification>
            <creationTime>2007-01-01T12:00:00.000+01:00</creationTime>
            <editTime>2007-01-01T12:00:00.000+01:00</editTime>
            <sealedMime>application/vnd.sealed.doc</sealedMime>
            <unsealedSize>1234567</unsealedSize>
        </contentDescription>
        <contentUri>http://server/files/fish.sdoc</contentUri>
        <account>
            <uuid>17f45d8d-d5c9-4970-8808-daa0fc893c33</uuid>
            <type>USER</type>
            <name>John Smith</name>
        </account>
    </core:DesktopState>
```

9 Reference

This section contains contains the following topics:

9.1 Terminology

The following table explains the main terms used in this Developer's Guide.

Table 9–1 Terminology

Term	Description
Classification System	A classification system describes a model for classifiying content. A classification system defines what metadata is sealed into content, how that metadata is used to grant access to that content and how cryptography is used with that content.
Classification	The set of metadata that tells the Oracle IRM Desktop what server to contact for rights, what cryptography keys to use to encrypt/decrypt/sign/verify the content, and the metadata used to associate rights with content (the classification cookie).
Classification Cookie	The set of metadata added to sealed content that is used by the Oracle IRM Desktop and IRM J2EE application to associate rights with content. The cookie is an opaque string of XML whose structure is defined by the classification system.
Public Header	The complete set of metadata added to sealed content, this includes the classification and custom data.
Custom Data	The set of metadata added to sealed content by third parties. This metadata is also signed and tamperproof, but the contents are ignored by the Oracle IRM products.
Sealed Content	Content that has been encrypted using Oracle IRM. Sealed content also contains signed metadata that is used by Oracle IRM.
Peeking	Peeking is the process of extracting the classification and custom data metadata from sealed content.
Sealing	Sealing is the process of taking content, adding metadata, signing this metadata and encrypting the content. The result of this transformation is called sealed content.
Unsealing	Unsealing is the process of taking sealed content and extracting the original, plaintext content. Unsealing can be considered the reverse process of sealing.

Table 9–1 (Cont.) Terminology

Term	Description
Resealing	Resealing is the process of altering the custom metadata or editing the encrypted content.
Reclassification	Reclassifying sealed content is the process of altering the classification of the sealed content. Reclassificating is typically used when content changes sensitivity, for example when a top secret document becomes a company confidential document.
Sealing Server	The sealing server is a J2EE application that allows sealed content to be processed remotely.
Context Journal	The context journal contains records of actions performed on context classified sealed content. The Oracle IRM Desktop maintains an audit of activity and uploads this to the IRM J2EE application. Context related activity is then stored in the context journal.

9.2 Feature Codes

The following feature codes can be used with the `Feature` type when creating or editing a `DocumentRole`.

Table 9–2 Feature codes

Feature	Description	Code
Open	Open and read a sealed file	oracle.irm.generic.Open
Seal	Create a new sealed file or seal an existing file	oracle.irm.generic.Seal
Reseal	Save changes to a sealed file	oracle.irm.generic.Reseal
Search	Search sealed files	oracle.irm.generic.Search
Copy	Copy the contents of a sealed file to the unprotected clipboard	oracle.irm.generic.Copy
Edit	Edit the contents of the sealed file and control change tracking	oracle.irm.generic.Edit
Print	Print the contents of a sealed file	oracle.irm.generic.Print
Print To File	Print the contents of a sealed file to a file or virtual print device, such as Acrobat	oracle.irm.generic.PrintToFile
Screen Capture	Capture the contents of a sealed file with 'Print Screen'	oracle.irm.generic.ScreenCapture
Set Item	Users are allowed to provide item codes when creating or saving sealed content. Without this option, sealed content is allocated an automatic item code	oracle.irm.generic.SetItem
Accessibility	Relaxes protection in sealed content to enable accessibility features to function fully	oracle.irm.generic.Accessibility
Copy To	Copy the contents of a sealed file to the sealed clipboard. The documents to which the content can be copied are configured separately	oracle.irm.generic.CopyTo
Save Unsealed	Save the contents of a sealed file into an unprotected file	oracle.irm.generic.SaveUnsealed

Table 9–2 (Cont.) Feature codes

Feature	Description	Code
Annotate	Add comments to sealed Word and Excel documents	oracle.irm.office.Annotate
Edit Tracked	Edit the contents of the sealed file with all changes tracked	oracle.irm.office.EditTracked
Interact	Enter data in form fields (Word) and unprotected cells (Excel)	oracle.irm.office.Interact
Formulae	View formulae (formulas)	oracle.irm.office.excel.Formulae
Reply	Edit the contents of sealed email and control change tracking	oracle.irm.office.email.Reply
Reply Tracked	Edit the contents of the sealed email with all changes tracked	oracle.irm.office.email.ReplyTracked
Program	Access content programmatically via the document object model	oracle.irm.office.Program

9.3 Locale Codes

The following locale codes are used in Oracle IRM.

Table 9–3 Locale Codes

Locale code	Language or language group
ar	Arabic
cs	Czech
da	Danish
de	German
el	Greek
en	English
es	Spanish
fi	Finnish
fr	French
hu	Hungarian
it	Italian
iw	Hebrew
ja	Japanese
ko	Korean
nl	Netherlands/Dutch
no	Norwegian
pl	Polish
pt-BR	Brazilian Portuguese
pt	Portuguese
ro	Romanian
ru	Russian

Table 9–3 (Cont.) Locale Codes

Locale code	Language or language group
sk	Slovak
sv	Swedish
th	Thai
tr	Turkish
zh-CN	Traditional Chinese
zh-TW	Simplified Chinese

Index

L

M

P

R

S

T

U

W