Videoscape Media Suite
Plug-in SDK Guide

For *Release 5.5*

Text Part Number: OL-30815-01

Notices

Trademark Acknowledgements

Cisco, Cisco Systems, the Cisco logo, and the Cisco Systems logo are registered trademarks or trademarks of Cisco Systems, Inc. and/or its affiliates in the U.S. and certain other countries.

All other trademarks shown are trademarks of their respective owners.

Publication Disclaimer

Cisco Systems, Inc. assumes no responsibility for errors or omissions that may appear in this publication. We reserve the right to change this publication at any time without notice. This document is not to be construed as conferring by implication, estoppel, or otherwise any license or right under any copyright or patent, whether or not the use of any information in this document employs an invention claimed in any existing or later issued patent.

Copyright

Table of Contents

Chapter 2

Overview

This chapter includes the following topics to introduce you to Media Suite plug-ins:

This guide gives Media Suite administrators an understanding of the software and related theory necessary to build and deploy custom VMS plug-ins. It is also intended as a reference for administrators to review specific plug-in details and examples that will assist in the creation of custom plug-ins

Understanding Media Suite

Media Suite is a modularized system that can facilitate all backend aspects of automated workflow processing, packaging, and the delivery of monetizable digital assets. It can provide all this for a diverse set of devices using a range of DRM technologies. The system is highly customizable, and supports a broad range of licensing, transcoding, encryption, and distribution models without the need to develop or redeploy new software. The default user interface tabs that you may have available within Media Suite are Workflow, Metadata, Entitlement, and Admin. If you purchase any additional optional modules, you will see those tabs as well.

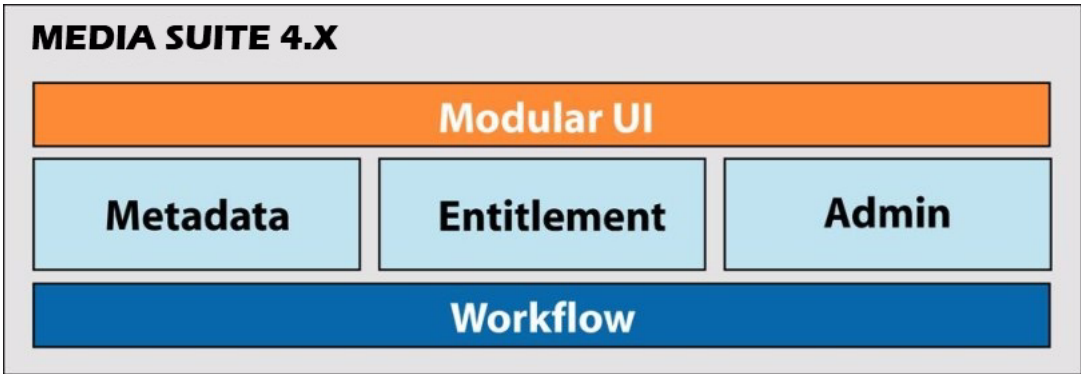
About Media Suite Modules

This section defines the various modules that are present within a default Media Suite installation. The modules are:

- Workflow
- Metadata
- Entitlement
- Admin

The following diagram shows the high-level relationship between various Media Suite modules:

Figure 1 Media Suite Modules at a High Level



Modular UI

Media Suite's user interface serves as a framework that is used by all core and any third party modules that may be configured with it.

Workflow Module

All functionality related to the creation and customization of workflows, and actions involved in those workflows, is managed through the Workflow module. In addition, the workflow module orchestrates all actions applied to content as it flows through Media Suite modules and any installed 3rd-party modules. To that end, this module also offers functionality for creating repository nodes and hot folders as well as for monitoring and intervening in workflows.

Metadata Module

The Metadata module is used to create or package metadata and physical assets into pre-defined objects such as components and bundles. This module is also where custom templates for bundles are created. Lastly, bind profiles, which match incoming content into bundle structures to automatically create bundles are managed through this module.

Entitlement Module

The Entitlement module's main purpose is to enable the creation of policies that specify business rules, and then to apply those policies to bundles in order to create products. Related functionality includes DRM setup, the ability to manage syndication, advertising, and users. Other module functions include the ability to manage currencies and feeds.

Admin Module

The Admin module is where administrators manage roles, administrators, and system-wide configuration settings.

About Media Suite Plug-ins

Videoscape Media Suite is a highly customizable product that allows users to implement plug-ins to create desired functionality. Each module contains a set of plug-in archetypes that can be expanded to create custom plug-ins that perform unique actions specified by the user.

Terminology

The following terminology tables have been organized into relevant areas to assist you in identifying what a particular Media Suite term pertains to. The terms are grouped with the following heading areas: general, binding, entitlement, and workflow.

General Terminology

The following table defines general Media Suite terms:

Table 1 General Media Suite Terminology

Term	Definition
ADMINISTRATOR	The people that use and configure Media Suite. The term “users”, on the other hand, refers to people that consume products that have been created by Media Suite.
AUTOMATION	All workflow processes are inherently automated within Media Suite and, once initiated, follow the logic that has been defined within the workflow.
BULK EDITING	Bulk edit functionality allows administrators to modify field values across multiple components at one time. This process overwrites previous values and cannot be undone, so care should be taken when bulk editing.
BUNDLE	Bundles are objects that conform to a template and are created when various components or other bundles are combined into a unit. New bundles can be customized. For further details on the default bundles that are available in Media Suite, see “Understanding Media Suite Bundles”.
BUNDLE TEMPLATE	Predefined templates used to establish and maintain a consistent structure for bundles. These templates include the minimum and maximum number of components that are allowed within a bundle. Once the minimum requirements are met, a bundle may be set to active.
COMMON ENTITY	A collection of normalized values that are reused throughout Media Suite. Default common entities include advisory, category, genre, rating, style, and talent (consisting of a person and a role.)
COMPONENT	Components are comprised of name-value pairs, common entities, and custom attributes that form the basic building blocks for bundles. A component’s structure cannot be edited via the Media Suite user interface, but custom attributes may be added to provide additional fields for the component.
CUSTOM ATTRIBUTE	Provides the ability to extend metadata for components by adding custom fields.
DOMAIN	Establishes a grouping of PCs and devices that can share licensed content within the group. Depending on the DRM technology capabilities, Media Suite can set registration limits for PCs or devices within individual or groups of DRM types.

Table 1 General Media Suite Terminology

Term	Definition
ENTERPRISE SERVICE BUS (ESB)	An Enterprise Service Bus is a software architecture construct that provides fundamental services for complex architectures. Within Media Suite, ESBs perform specific functions that are configured within action templates and then further detailed within action profiles.
LOCALE	A locale refers to specific language and regional settings, such as number, date, and currency formats. Locales are specified using a language code and a country code, such as en_US.
OPENCASE	Versions of Media Suite up to and including 4.0 were branded as OpenCASE and were developed by ExtendMedia Inc., which was acquired by Cisco Systems Inc. in September 2010.
USER	The people that consume products that have been generated by Media Suite.

Binding Terminology

The following table defines terminology that is related to Media Suite binding:

Table 2 Binding-specific Terminology

Term	Definition
ATTACHMENT RULES	A set of rules on a destination folder that direct the binding of components to folders within a bundle.
BINDING	A process that automatically creates a bundle from a diverse set of incoming content.
BUNDLE ASSOCIATION	A set of rules that helps Media Suite determine which incoming objects should be associated as belonging to the same bundle.
COMPONENT ASSOCIATION	A set of rules that helps Media Suite determine where incoming content should be directed on the bundle tree structure.
FOLDER ASSOCIATION	A set of rules that helps Media Suite determine where incoming content should be directed when a parent folder contains multiple versions of the same object type.

Entitlement Terminology

The following table defines terms that are specific to Media Suite entitlement functionality:

Table 3 Entitlement-specific Terminology

Term	Definition
POLICY	Policies are a mechanism to establish a set of business rules that will be applied to bundles to create a product. A policy encompasses product aspects such as offer windows, offer types, DRM characteristics, ad support, and charges.
PRODUCTIZATION	Is the process of applying a policy to a bundle to create a product.
RIGHTS LOCKER	A centralized repository for rights tokens for a user account. Rights tokens grant users the permission to purchase or obtain a product.

Workflow Terminology

The following table defines terms that are specific to Media Suite workflow functionality:

Table 4 Workflow-specific Terminology

Term	Definition
Action Profile	A configured instance of an action template. An action template may specify that a transcode be performed while an action profile would further specify to transcode using Expression Encoder.
Action Template	A generic action that processes, transforms, or transports a file. Action templates establish a generic workflow action such as Transcode.
Cisco Transcode Manager (CTM)	Cisco Transcode Manager is the new name for what was previously called Armada. The product is an enterprise-class transcode workflow solution that eases the complexities of encoding large volumes of content. At present, references to Armada still exist within ESBs and various Media Suite code.
ContentFile	ContentFiles track the path of files as they move through a workflow.
Filesystem	Filesystems are mounted so that files may be browsed by Media Suite.
Hot Folder	A location on a file system that has been designated as a drop point for physical files. Hot folders are polled at frequent intervals.
Repository	The collection all repository nodes within Media Suite.
Repository Node	Repository nodes are created by mounting a filesystem. They may (optionally) have one or more hot folders that are linked to the filesystem.
Task Monitor	A place where administrators can view active, suspended, or completed tasks that have been assigned to them.
Transcoding	Is the digital-to-digital conversion of one video encoding to another. Common encoding formats that can be used within Media Suite include: HTTP Live Streaming (HLS), which is an Apple format Smooth Streaming, which is a Microsoft format Multi-Bitrate, which is an Adobe format
Workflow	A collection of steps or decision branches that may span one or more Media Suite modules or 3rd party applications.
Workflow Instance	Any time a workflow starts (for each file) it creates a unique workflow instance.
Workflow Instance Monitor	A means for administrators to review workflow activity using various metrics. The workflow monitor is also used for suspending, resuming, ending, and deleting workflows.
Workflow Node	Within the graphical workflow definition view, this is a specific point where an action is intended to be performed. Each node exposes a set of configuration options within the Media Suite interface. A configured node example would be “Encrypt using PlayReady”.
Workflow Definition	Is a configured instance of a workflow template. It articulates the steps and decision branches

Table 4 Workflow-specific Terminology

Term	Definition
WORKFLOW TASK	The point at which a workflow has to stop and wait for a decision by an administrator. The status of tasks is managed through the task monitor in Media Suite.
WORKFLOW TEMPLATE	Is an imported Process Definition (PAR file) that was created within a visual process design tool. Workflow templates articulate steps and decision pathways within a workflow. These templates are a starting point to model customized workflows for a deployment.

EPG & Search Manager Terminology

Table 5 EPG & Search Manager-specific Terminology

Term	Definition
CHANNEL	Channels are stations that have been assigned a channel number and are mapped to a channel line-up.
CHANNEL LINE-UP	Channel Line-ups are a set of channel mappings created for each service provider's geographical or virtual region. Channel line-ups may be further segregated by device type.
CHANNEL MAP	Channel maps are when a station is assigned to channel number within a line-up. For example, assigning the station NBC to channel 91 within the Rogers Toronto HD line-up.
EPG	Electronic Program Guides are used to deliver TV scheduling information for a given period of time to an end user. EPGs are targeted to various devices, such as set-top boxes (STBs), smartphones, tablets, and PCs.
EXTERNAL STATION	Are used as basis to create channel line-ups. They are managed external to Media Suite and must be imported via a feed. External station are synonymous with "stations".
FEED	In our context, this is electronic program guide data (XML or otherwise) that is ingested into the EPG module.
FEED FRAGMENT	Feed fragments are a portion of a feed that holds a particular aspect of the complete feed. Feed fragments may include data such as schedules, program information, stations, line-ups, or regions. Custom feed fragment types may also be specified within Media Suite.
IM	Linear Manager (lm) was an earlier name for EPG that was used during design and development. You may see references to lm in the source code and in the System Configuration parameters in the Media Suite user interface.
INDEX MASTER	Is synonymous with the Solr Master (see definition below).
INGEST REGION	Ingests a TV Schedule feed fragment that contains generic information for the geographic regions, such as zip/postal codes or other identifiers.
MOCK	The process of creating special case objects that mimic real objects for testing purposes.
PROGRAM	Content metadata managed by the EPG module. This may include movie, television, or radio shows.
REGION	A region is synonymous with a channel line-up (see definition above).
SCHEDULE	A schedule (or event) is a single program occurring on a specific station at a specific time.

Table 5 EPG & Search Manager-specific Terminology

Term	Definition
SEARCH CONFIG MANAGER	Search config manager is a component that enables EPG module configuration from the Media Suite user interface.
SEARCH MANAGER	Search Manager enables searches to be performed on the indices provided by the Solr Slave.
SOLR	See “Understanding Solr Tech
SOLR MASTER	A Solr Master generates and updates its own indexes and makes them available to one or more SOLR slaves.
SOLR SLAVE	Solr Slaves extract information for the Solr Master and then make it available to clients for consumption.
STATION	Stations are identified by call signs (such as CTV or NBC). A station becomes a channel once it has been assigned a channel number (such as 9) and mapped into a specific channel line-up. Stations are synonymous with “external stations”.
VOD	Although this term specifically applies to Video On Demand content, what is applicable to VoD is also applicable to other on-demand content, such as audio tracks, ebooks, and any digital media available through Content Manager.

Chapter 3

Getting Started

This chapter includes the following topics related to creating and deploying Media Suite plug-ins:

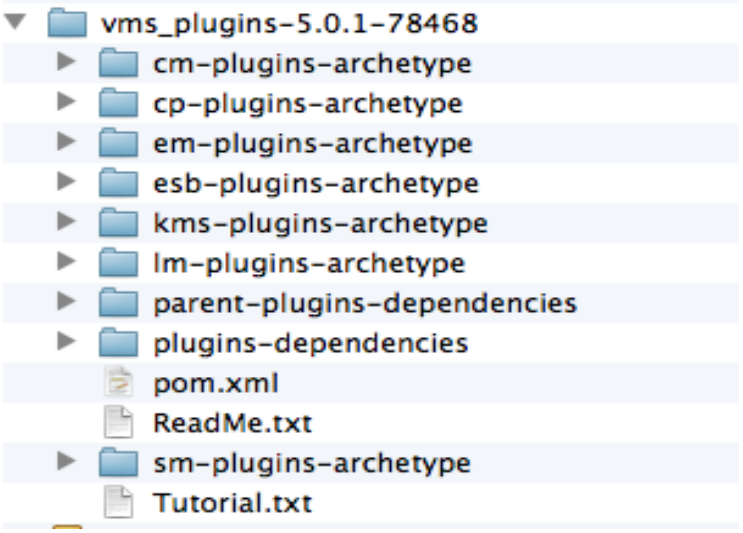
- "Required Software", as shown below
- “Preparing a Workspace” on page 20
- “Building a Signed or Unsigned Plug-in” on page 20
- “Deploying a Plug-in to the VMS Server” on page 21

Required Software

Table 6 Required Software for Creating Custom Plug-ins

Software	Use
MAVEN	The build process for custom VMS plug-ins is based in Maven; this software is necessary to successfully build a custom plug-in. We recommend version 3.0.5.
JDK	JDK is the base library for all plug-ins. We recommend version 1.7 when using VMS 5.5 and version 1.6.0_38 for earlier VMS versions.
ECLIPSE	Eclipse is an optional tool; users who choose to work in Eclipse should install with the JBoss tools add-on.

Folder Structure - Image should be changed for 5.5



Preparing a Workspace

Complete the following steps to prepare your workspace.

1. Install Maven. To install Maven, visit http://maven.apache.org/run-maven/index.html and follow the instructions given.
2. Open command line.
3. Execute: `cd <plugins parent directory>`
4. Execute: `<mvn clean install>` to generate the plug-in archetypes.
5. Create a project folder titled "`<my project>`"
6. Execute: `cd <my project>`
7. Execute the following to create a skeleton for custom plug-ins:

```
mvn archetype:generate
-DarchetypeGroupId=<archetype-groupId)>
-DarchetypeArtifactId=<archetype-artifactId)>
-DarchetypeVersion=<build version)>
-DgroupId=<my groupId)>
-DartifactId=<my-artifactId)>
-Dversion=<custom version>
```

8. Execute: cd <my project/custom-xxx-plugin>

Building a Signed or Unsigned Plug-in

Complete the following steps to build either a signed or unsigned plug-in.

1. Building a signed plug-in:

a. Configure keystore properties in: `<my-project>/custom-xxx-plugin/src/main/resources/pom.propeteries.`

b. Execute:

```
cd <my-project>/custom-xxx-plugin>
mvn -DskipJarSigning=false clean install
```

Note To merge any dependencies missing in JBoss into JAR with the created plug-in, update: `<my-project-root>/<artifactID>/src/main/resources/assembly-descriptor.xml`. Include all dependencies that you want to merge in the `<includes>` section. The `<includes>` section contains a sample of how to include dependencies. After the assembly-descriptor.xml file is updated, invoking '`mvn install`' will build the JAR and include all configured dependencies into this JAR.

---

2. Building an unsigned plug-in:

   a. Run the following command:

   ```
   mvn clean install
   ```

Deploying a Plug-in to the VMS Server

1. Deploy the following file into the VMS server.
   `<my-project-root>/<artifactId>/target/<my-artifactId>-<my-version-for-custom-plugin-jar>.jar`

   a. Navigate to the **Configure** tab in the Installer Manager User Interface.

   b. Select **Custom File Deployment** and add a new custom file.

   c. Select the target module(s) required for the plug-in.

   d. Place the file inside the module directory and update the target path to the `lib` folder.

   e. Upload and save.

   f. Activate the plug-in

Note Ensure the `.jar` file is successfully deployed to the VMS directory through the agent at `${JBOSS_HOME}/${JBOSS_DOMAIN}/deploy/${Modules.ear}/lib`. For example: `/opt/cisco/vms/var/jboss/server/vms/deploy/ContentManager.ear/lib`.

---

   g. Restart VMS.

Chapter 4

Creating Custom Content Manager Plug-ins

This chapter includes information about creating the following custom Content Manager plug-ins:

Content Manager

Prerequisites

The following steps are applicable to each of the Content Manager plug-ins listed below. These build and deploy steps must be performed before configuring or triggering the plug-ins.

1. Install the parent POM file into the local Maven repository.
2. Generate the plug-in archetype.
3. Create a "Project Root" folder on a local computer to house the custom plug-in(s).
4. Build the plug-in files and place in the "Project Root" folder.

```
cd <Project root>
mvn archetype:generate
-DarchetypeGroupId=com.xxx.vms.plugins
-DarchetypeArtifactId=vms-cm-plugins-archetype
-DarchetypeVersion=x.x-xxxxxx (plugins version)
-DgroupId=com.xxx.vms
-DartifactId=custom-cm-plugin
-Dversion=1.25
```

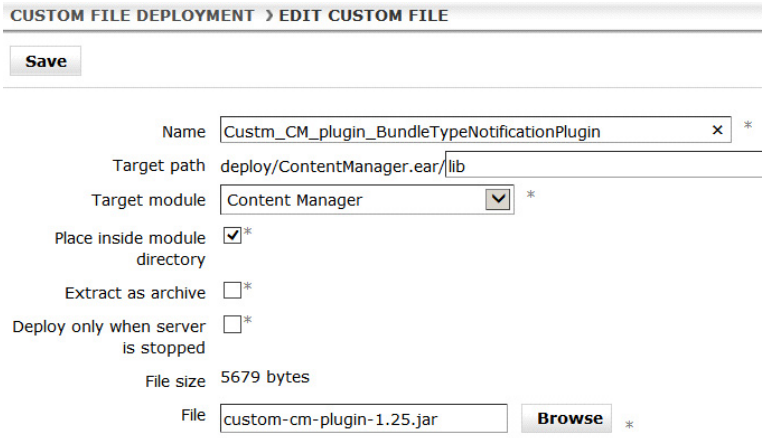
5. Build a signed or unsigned JAR file.
6. Deploy the JAR file into the VMS server.
   a. Navigate to **Configure > Custom File Deployment > Add New**.
   b. Enter a name.
   c. Set the target path to: deploy/ContentManager.ear/lib.

d Set the target module to: Content Manager.

e Check the box “Place inside module directory”.

f Browse and select the created JAR file.

g Click **Save** then **Activate**.



7 Restart VMS.

Entity Change Queue Plug-in

Plug-in Architecture

To view the plug-in architecture, see the attached Plug-in Architecture zip file.

Description

The Entity Change Queue plug-in is used to notify other applications of changes made to common entities, products, bundles and components. This plug-in handles a set of object changes and will be invoked in the case of a bundle or product change (but not a feed or policy change). It removes duplication of product notifications and creates a single row on the entity change queue table for the specific object.

To view the methods this interfaces specifies, open the Content Manager folder in “vms_docs-[releasenumber]-[buildnumber]” and select “index.html”.

Primary Use Case:

The primary use of the Entity Change Queue plug-in is to populate feeds with notifications related to entity changes.

Example

The following is a sample Entity Change Queue plug-in.

```
public class SampleEntityChangeQueuePlugin implements EntityChangeQueuePlugin {
```

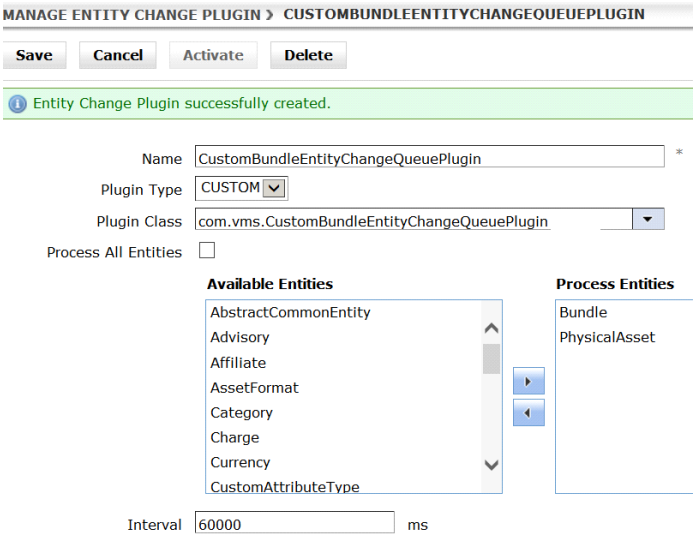
```
	private static final transient Log LOG = LogFactory.getLog(SampleEntityChangeQueuePlugin.class);

	/**
	 * Executes the custom Entity change plug-in's logic on the specified object.
	 *
	 * @param className Class name for the object in the queue
	 * @param objectPrimaryKey Primary key for the object in the queue
	 * @param plugin The entity change plug-in processing the object.
	 * @param action that specifies how object has changed
	 * @param entityChangeQueuePk reference to entity change queue item containing objectPrimaryKey, plug-in and
	        action
	 *
	 * @throws EntityChangePluginException if an error occurs when processing an entity change queue item. This will
	indicate to the entity change queue job that the entity change queue item should not be deleted and retried later.
	 */
	@Override
	public void process(String className, Integer objectPrimaryKey, EntityChangePlugin plugin, LifecycleActionsEnum
	action,
	        Integer entityChangeQueuePk) throws EntityChangePluginException {

	        //Retrieve rest webservice
	        BundleWebservice webservice = (BundleWebservice) Component.getInstance(BundleWebservice.class, true);

	        Class<?> clazz = null;
	        try {
	                clazz = Class.forName(className);
	        } catch (ClassNotFoundException e) {
	                LOG.warn("process() Class for object [{" + className + "}] was not found.");
	                return;

	        //Only bundle object
	        if (Bundle.class.equals(clazz)) {
	                LOG.info("SampleEntityChangeQueuePlugin invoked [" + className + "], PK [" + objectPrimaryKey
	                + "], plugin name ["
	                                + plugin.getName() + "], Action [" + action.toString() + "],
	                                EntityChangeQueuePk [" + entityChangeQueuePk + "].");

	                switch (action) {
	                //create operation for bundle
	                case CREATE:
	                        LOG.info("Bundle create notification received for objectPrimaryKey [" +
	                        objectPrimaryKey + "].");
	                        break;

	                //update(modify) operation for bundle
	                case UPDATE:
	                        String bundleXml = null;
	                        try {
	                        bundleXml = webservice.getBundleAsXMLByComponentPrimaryKey(objectPrimaryKey);
	                        } catch (MarshallerException e) {
	                                LOG.error("Unable to marshall bundle.", e);
	                                return;
	                        } catch (BundleWebserviceException e) {
	                                LOG.error("Unable to retrieve bundle from webservice.", e);
	                                return;
	                        }
	                        LOG.info("Bundle update notification received for bundle xml [" + bundleXml + "],
	                                Action [" + action.toString()+ "], EntityChangeQueuePk [" +
	                                entityChangeQueuePk +"].");
	                        break;

	                //delete operation for bundle
	                case DELETE:
	                        LOG.info("Bundle delete notification received for objectPrimaryKey [" +
	                        objectPrimaryKey + "].");
	                        break;
	                }
	        }
	        return;
	}
}
```

Configuring the Plug-in

Perform the following steps to configure a custom Entity Change Queue plug-in:

1 Add a new Entity Change Plug-in.

- **a** Navigate to **Entitlement > Setup > Plugins > Entity Change Plugins > Add New**.
- **b** Enter a name.
- **c** Set the Plugin Type to: Custom
- **d** Set the Plugin Class to: CustomEntityChangeQueuePlugin.
- **e** Select "Bundle" from the **Available Entities** list.
- **f** Click **Save** then **Activate**.



Triggering the Sample Plug-in

2 Create a new bundle.

- **a** Navigate to **Metadata > Bundles > Add New.**
- **b** Enter a name.
- **c** Click **Save** & **Edit**.

Expected Results

This sample will log the CRUD operation of a bundle. If it is an update, the bundle XML is also logged.

Abstract Bundle Change Plug-in

Description The Abstract Bundle Change plug-in is an implementation of the Entity Change Queue plug-in. It is used to validate bundles and to publish bundle change data to an external system.

This plug-in will: take an entry from the bundle change queue, get appropriate bundle data, validate the data, and call abstract methods that should be implemented by subclasses that export data to a specific system.

To view the methods this interfaces specifies, open the Content Manager folder in "vms_docs-[releasenumber]-[buildnumber]" and select "index.html".

VCM Bundle Change Plug-in

Description The VCM Bundle Change plug-in extends the AbstractBundleChangePlugin and is a specific Merchandiser EntityChangeQueue plug-in. It is responsible for publishing bundle change data to the Merchandiser system. This plug-in takes an entry from the bundle change queue, gets appropriate bundle data, validates the data, and calls abstract methods that should be implemented by subclasses which export data to a specific system.

To view the methods this interfaces specifies, open the Content Manager folder in "vms_docs-[releasenumber]-[buildnumber]" and select "index.html".

Primary Use Case:

The primary use case is to synchronize VMS metadata and asset data with the Merchandiser system.

Abstract Integration Bundle Change Plug-in

Description TThe Abstract Integration Bundle Change plug-in extends the Abstract Bundle Change plug-in. It creates a base class that will be used to validate contents of a bundle after a change and then notify any external systems of relevant changes. Any class that extends this plug-in will also need to provide a validator class and a transaction processor class. Validator classes will extend "abstract integration abstract plugin validator" and the transaction process will extend "abstract integration bundle change transaction processor".

To view the methods this interfaces specifies, open the Content Manager folder in the Javadocs and select "index.html".

Primary Use Case:

This plug-in is primarily used to integrate Videoscape Media Suite with external systems such as VBO, VCM, and Concurrent.

Content Manager Notification Plug-in

Plug-in Architecture

To view the plug-in architecture, see the attached Plug-in Architecture zip file.

Description

The Content Manager Notification plug-in creates a notification for changes that occur in: bundles, components, common entities, products, affiliates, entitlement profiles, charges, currencies, DRM profiles, feeds, policies, subscription codes, origin mappings, retention policies, custom attributes, and WizardTemplateConfig. The plug-ins used to extend this class will process the object's notification and create an EntityChangeQueue entry.

To view the methods this interfaces specifies, open the Content Manager folder in "vms_docs-[releasenumber]-[buildnumber]" and select "index.html".

Primary Use Case:

The primary use of the Notification plug-in is to populate the entity change queue table with the associated entity change queue plug-ins.

Example

The following is a sample Content Manager Notification plug-in.

```
/**
 * Notification entry will be processed by the custom logic
 *
 * @param object the entity which the event occurred.
 * @param objectPrimaryKey The primary key of the object the notification is triggered for.
 * @param action the life cycle event that occurred.
 * @param notificationData Any optional data associated with the type of object the notification is triggered by.
 * @param notificationCreateDate Creation date of notification.
 * @throws NotificationPluginException If an error occurs when processing a notification. This will indicate to the
notification job that the notification should not be deleted and retried later.
 */
public class SampleNotificationPlugin implements NotificationPlugin {

         private static final transient Log LOG = LogFactory
                           .getLog(SampleNotificationPlugin.class);

         @Override
         public void processNotification(String object, Integer objectPrimaryKey,
                           String action, String notificationData,//
                           Date notificationCreateDate) throws NotificationPluginException {

                  try {
                           final Class<?> clazz = Class.forName(object);
                           final LifecycleActionsEnum actionEnum = LifecycleActionsEnum
                                             .valueOf(action);

                           // Only bundle class will be handled by this plugin
                           if (clazz.isInstance(new Bundle())) {

                                    LOG.info("Invoked Custom Plugin SampleNotificationPlugin with object ["
                                                      + object
                                                      + "], PK ["
                                                      + objectPrimaryKey
                                                      + "], notificationData ["
                                                      + notificationData
                                                      + "], Action ["
                                                      + action.toString()
                                                      + "], notificationCreateDate ["
                                                      + notificationCreateDate.toString() + "].");

                                    if (LifecycleActionsEnum.CREATE.equals(actionEnum)) {
                                             LOG.info("Bundle item found with action create");

                                    //Use the component webservice to retrieve the bundle type
                                             String bundleTypeName = getComponentWebservice()
```

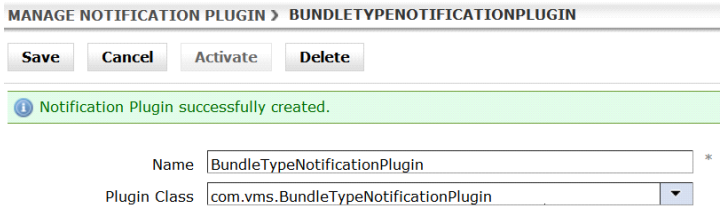
```
                                                        .findBundleTypeNameByBundleId(objectPrimaryKey);
                                                LOG.info("The created bundle is a [" + bundleTypeName
                                                        + "].");
                                        }
                                }
                        } catch (Exception e) {
                                throw new NotificationPluginException(
                                        "Failed to process Notification", e);
                        }
                        return;
                }


                // Retrive the local interface for webservice
                protected ComponentWebServiceLocal getComponentWebservice() {
                        return (ComponentWebServiceLocal) Component.getInstance(ComponentWebService.class);
}
```

Configuring the Plug-in

Perform the following steps to configure a custom Notification plug-in:

1 Add a new Notification plug-in.

- **a** Navigate to **Entitlement > Setup > Plugins > Notification Plugin > Add New.**
- **b** Enter a name.
- **c** Click C**reate & Edit**.
- **d** Select a plug-in class.
- **e** Click **Save** then **Activate**.



Triggering the Sample Plug-in

2 Create a new bundle.

- **a** Navigate to **Metadata > Bundles > Add New > Logical Video.**
- **b** Enter a name.
- **c** Click **Create & Edit.**
- **d** Click **Save** then **Activate**.

Expected Results

This sample will log the bundle type when a bundle is created.

Component Update Plug-in (Bind Profile)

Plug-in Architecture

To view the plug-in architecture, see the attached Plug-in Architecture zip file.

Description

The Component Update plug-in is an interface used by the bind profile plug-in to update a component during the bind process. Default behavior is to merge components during the bind process.

To view the methods this interfaces specifies, open the Content Manager folder in “vms_docs-[releasenumber]-[buildnumber]” and select “index.html”.

Primary Use Case:

The primary use case for this plug-in is to merge specific component types during the bind process.

Example

The following is a sample Component Update plug-in.

```
public class SampleComponentUpdatePlugin extends PhysicalAssetUpdatePlugin implements ComponentUpdatePlugin {

        private static final transient Log LOG = LogFactory.getLog(SampleComponentUpdatePlugin.class);

        /**
         * This method decides which component can be handled by this plugin.
         * For this sample, only image is handled
         *
         * @param component The incoming component
         * @return true if the component can be handled
         */
        @Override
        public boolean isHandled(AbstractComponent component) {
                if(component == null) {
                        return false;
                }
                LOG.info("Check if SampleComponentUpdatePlugin can handle component [" + component + "]");

                return (component instanceof PhysicalAssetImage);
        }

        /**
         * This method dictates how to find the existing image that will be overriding the physical asset.
         * This can be customized to fit how the name of the file can be used to match to certain field of the asset in
         * order for the update to trigger
         *
         * @param incomingComponent component to merge with
         * @return Returns true if a component was successfully found, merged and persisted to database. Returns false if
                component in bundle could not be found based on criteria.
         * @throws BindException if components are null, not of the same type or not handled by plugin
         * @throws ComponentManagerException if merged component could not be persisted to database
         */
        @Override
        public boolean findAndMergeComponent(AbstractComponent incomingComponent, String originalSourceFilePath)
                        throws BindException, ComponentManagerException {
                LOG.info("SampleComponentUpdatePlugin, findAndMergeComponent executed with AbstractComponenet [" +
                incomingComponent.getName() +
                                "], originalSourceFilePath [" + originalSourceFilePath + "].");

                AbstractComponent existingComponent = null;
```

```
        try {
                existingComponent = findMatch(originalSourceFilePath);
                if(existingComponent == null) {
                        LOG.warn("Unable to find matching component to incomingComponent [" +
                        incomingComponent.getComponentPk() + "]");
                        return false;
                } else {
                        LOG.info("Found matching component [" + existingComponent.getComponentPk() + "] to
                        incomingComponent ["
                                        + incomingComponent.getComponentPk() + "]");
                }
        } catch (ComponentParameterFault e) {
                LOG.error("Unable to find component with externalId [" + originalSourceFilePath + "].");
        } catch (ComponentWebserviceFault e) {
                LOG.error("Unable to find component with externalId [" + originalSourceFilePath + "].");
        }

        // Temporary copy used to avoid constraint violation and situations where a delete work flow can be
        triggered when merging components.
        String contentId = ((PhysicalAssetImage) incomingComponent).getContentID();
        String contentFileUuid = ((PhysicalAssetImage) incomingComponent).getContentFileUuid();
        String deleteUrl = ((PhysicalAssetImage) incomingComponent).getOriginDeleteUrl();
        LOG.info("Saved content ID [" + contentId + "]; content file UUID [" + contentFileUuid + "]; delete URL
        [" + deleteUrl + "].");

        // Clear fields that may cause unique constraint violations.
        ((PhysicalAssetImage) incomingComponent).setContentID(null);
        ((PhysicalAssetImage) incomingComponent).setContentFileUuid(null);
        ((PhysicalAssetImage) incomingComponent).setOriginDeleteUrl(null);
        incomingComponent.setBindRequest(null);

        try {
                getComponentWebservice().updateComponent(incomingComponent);
        } catch (ComponentParameterFault e) {
                LOG.error("Unable to update component with id [" + incomingComponent.getUuid() + "].");
        } catch (ComponentWebserviceFault e) {
                LOG.error("Unable to update component with id [" + incomingComponent.getUuid() + "].");
        }
        AbstractComponent mergedComponent = mergeComponents(existingComponent, incomingComponent, false);

        ((PhysicalAssetImage) mergedComponent).setContentID(contentId);
        ((PhysicalAssetImage) mergedComponent).setContentFileUuid(contentFileUuid);
        ((PhysicalAssetImage) mergedComponent).setOriginDeleteUrl(deleteUrl);
        try {
                //Update the existing physical asset
                getComponentWebservice().updateComponent(mergedComponent);
        } catch (ComponentParameterFault e) {
                LOG.error("Unable to update component with id [" + mergedComponent.getUuid() + "].");
        } catch (ComponentWebserviceFault e) {
                LOG.error("Unable to update component with id [" + mergedComponent.getUuid() + "].");
        }

        return true;
}

/**
 * This sample finds matching components based on external id. Existing component is expected to have the filename
 of image as external ID.
 * (ExternalId = FileName.ext), this can be customized however you need.
 *
 * @param originalSourceFilePath the source file path and name
 * @return Returns component with matching criteria. Returns null if match not found.
 *
 * @throws ComponentParameterFault
 * @throws ComponentWebserviceFault
 */
protected AbstractComponent findMatch(String originalSourceFilePath) throws ComponentParameterFault,
ComponentWebserviceFault {
        LOG.info("Attempting to find a matching component to incoming image with original source file path [" +
        originalSourceFilePath + "]");
        return getComponentWebservice().findComponentByExternalID(originalSourceFilePath);
}
```

```
        /**
         * Merge an existing PhysicalAssetImage component with a new copy of PhysicalAssetImage from a component update
         * The attributes that will be merged can be controlled
         *
         * @return merged component
         */
        protected AbstractComponent mergeImage(PhysicalAssetImage existingComponent, PhysicalAssetImage
        incomingComponent)
                        throws BindException {
                LOG.info("SampleComponentUpdatePlugin mergeImage invoked for [" + existingComponent.getName() +
                                "] and [" + incomingComponent.getName() + "]");

                existingComponent.setRepositoryIngestFilePath(incomingComponent.getRepositoryIngestFilePath());
                existingComponent.setFileSize(incomingComponent.getFileSize());
                existingComponent.setCheckSum(incomingComponent.getCheckSum());
                existingComponent.setMimeType(incomingComponent.getMimeType());
                existingComponent.setPcFileName(incomingComponent.getPcFileName());
                existingComponent.setOriginBaseUrl(incomingComponent.getOriginBaseUrl());
                existingComponent.setOriginResourceUrl(incomingComponent.getOriginResourceUrl());
                existingComponent.setHeight(incomingComponent.getHeight());
                existingComponent.setWidth(incomingComponent.getWidth());
                existingComponent.setTokenizedUrl(incomingComponent.getTokenizedUrl());
                existingComponent.setCreateDate(incomingComponent.getCreateDate());

                return existingComponent;
        }

        protected ComponentWebServiceLocal getComponentWebservice() {
                return (ComponentWebServiceLocal)Component.getInstance(ComponentWebService.class);
        }
}
```

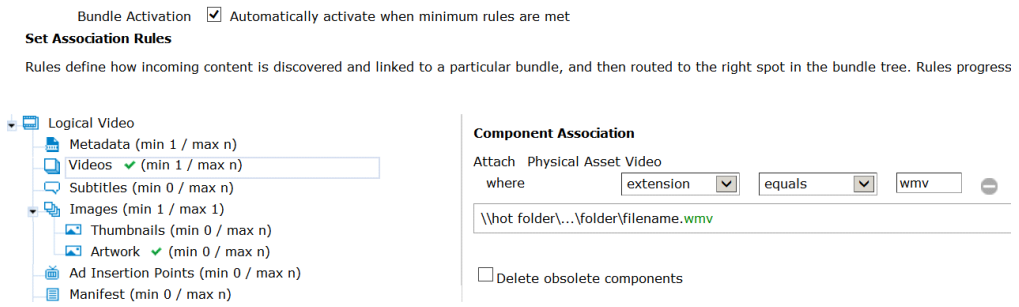
Configuring the Plug-in

Perform the following steps to configure a custom Component Update plug-in:

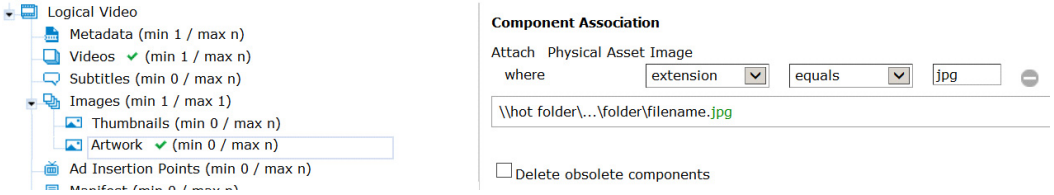
1. Open VMS.
2. Add a new Component Update plug-in.
   a. Navigate to **Metadata > Setup > Bind Profile Plugins > Add New.**
   b. Enter a name.
   c. Click **Create & Edit.**
   d. Set the Plugin Class to: com.custom.plugin.SampleComponentUpdatePlugin
   e. Click **Save** then **Activate.**

Triggering the Sample Plug-in

3. Create a new bind profile.
   a. Navigate to **Metadata > Bind Profiles > Add New.**
   b. Enter a name.
   c. Select “Bind Template” from the drop-down menu.
   d. Click **Create & Edit**.
4. For videos, set the following values:

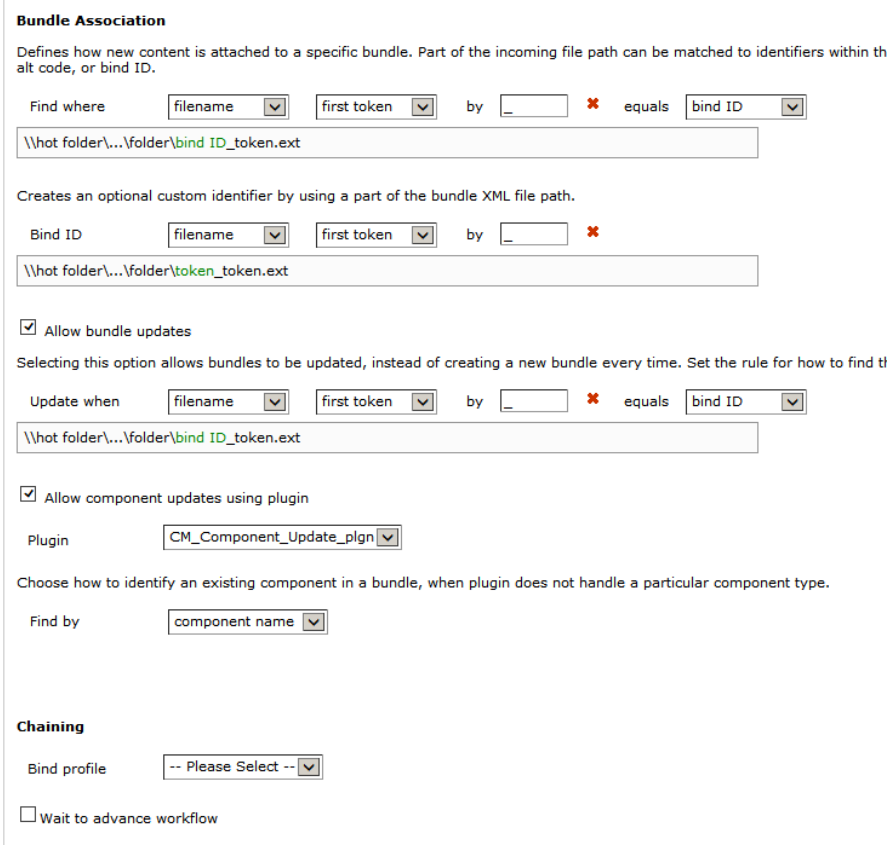


5 For images, set the following values:



6 Click **Save** then **Activate**.

7 For logical videos, set the following values:

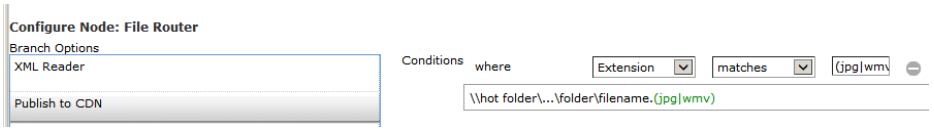


8 Click **Save** then **Activate.**

9 Create a Workflow Definition.

a In the **Configure** tab, set the following values for XML Reader:

b In the **Configure** tab, set the following values for Publish to CDN:

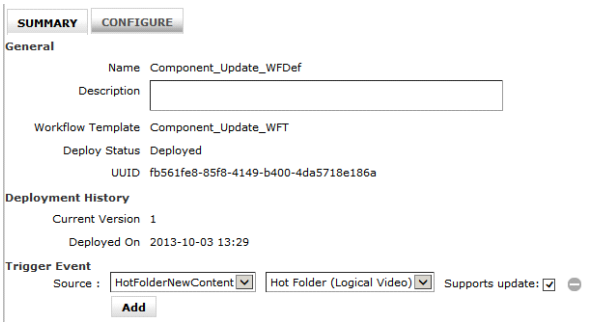


c Build Metadata by selecting the Bundle Profile from the **Processor** drop-down menu.

d Productize by selecting the policy from the **Policy** drop-down menu.

e Publish to CDN by selecting action profile from the **Processor** drop-down menu.

f In the **Summary** tab, set the following values:



g Click **Save**.

h Click **Deploy**.

i Click **Activate.**

10 Place `.xml, .jpg and .wmv` files into the Hot Folder/Logical Video/In folder.

11 Check that the correct bundle was created after placing the files.

a Navigate to **Metadata > Bundles > Search.**

12 Check that the image was added.

a Navigate to **Metadata > Bundles > Search > Logical Image.**

b Select the logical image.

c Check that the image file is located under **Images > Artwork.**

d Select the image and click **Remove.**

e Click **Add New**.

f Enter a new name.

g Set the External ID as the full name of the `.jpg` file. For example, MyLogicalImage.jpg will be used as the External ID.

h Click **Save**.

i Delete the `.jpg` file from the Hot Folder/Logical Video/In folder.

j Re-add the `.jpg` file.

k Run the workflow.

Expected Results

This sample merges a new physical asset image with an existing image based on the ExternalId and filename.

Chapter 5

Creating Custom Content Processor Plug-ins

This chapter includes information about creating the following custom Content Processor plug-ins:

Content Processor

Path Processor Plug-in

Plug-in Architecture

To view the plug-in architecture, see the attached Plug-in Architecture zip file.

Description

Implementations of the Path Processor plug-in are invoked in copy and move Repository Manager operations through a workflow. The plug-in is intended to allow the implementer to transform the system-generated output path for these operations.

To view the methods this interfaces specifies, open the Content Processor folder in "vms_docs-[releasenumber]-[buildnumber]" and select "index.html".

Primary Use Case:

This plug-in is used when images stored in source location do not match intended paths on the destination server. This plug-in can provide translation service to alter the paths and file names to match the required naming scheme.

Example

The following is a sample Path Processor plug-in.

```
/**
 * This is a sample of PathProcessorPlugin
 *
 * This plugin allows customization of file operation. This example will copy/move the file into a time stamped folder.
 *
 *  Note: This plugin has to be declared as a Seam component and with a @Name annotation
 */
@Name("SamplePathProcessorPlugin")
public class SamplePathProcessorPlugin implements PathProcessorPlugin {
```

```
        private static final Log LOG = LogFactory.getLog(SamplePathProcessorPlugin.class);

        public SamplePathProcessorPlugin() {
                super();
        }

        /**
         * Returns the path of the destination, in this case, a folder with the time stamp will be created to store the
         file
         * destPath is in the format of path (i.e. /opt/test/destionfolder/)
         */
        public String processPath(String sourcePath, String destPath) {
                LOG.info(String.format("SamplePathProcessorPlugin, process path invoked with source path [%s] and
                destination path [%s]", sourcePath, destPath));
                Date currentDate = new Date();
                String destinationPath = destPath + currentDate.toString();
                return destinationPath;
        }
}
```

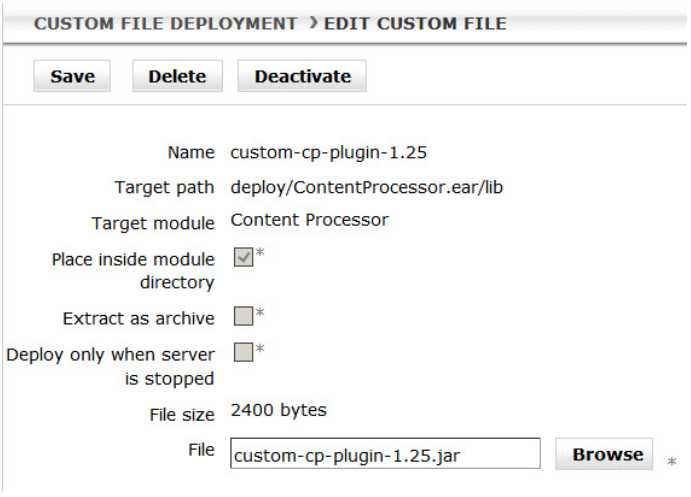
Configuring the Plug-in

Perform the following steps to configure a custom Path Processor plug-in:

1. Install the parent POM file into the local Maven repository.
2. Generate the plug-in archetype.
3. Create a “Project Root” folder on a local computer to house the custom plug-in(s).
4. Build the plug-in files and place in the “Project Root” folder.

```
cd <Project root>
mvn archetype:generate
-DarchetypeGroupId=com.xxx.vms.plugins
-DarchetypeArtifactId=vms-cp-plugins-archetype
-DarchetypeVersion=x.x-xxxxxx (plugins version)
-DgroupId=com.xxx.vms
-DartifactId=custom-cm-plugin
-Dversion=1.25
```

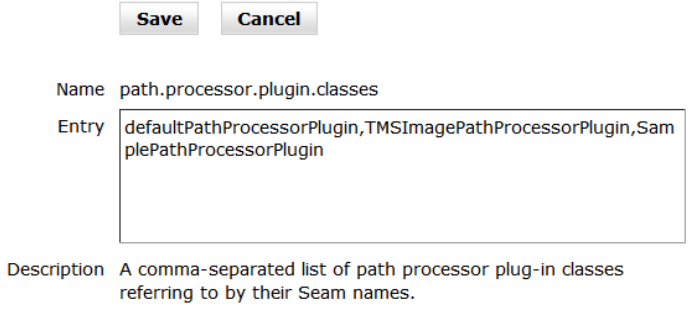
5. Build a signed or unsigned JAR file.
6. Deploy the JAR file to the VMS server.
   a. Navigate to **Configure > Custom File Deployment > Add New.**
   b. Enter a name.
   c. Set the target path to: deploy/ContentProcessor.ear/lib.
   d. Set the target module to: Content Processor.
   e. Check the box “Place inside module directory”.
   f. Browse and select the created JAR file.
   g. Click **Save** then **Activate.**



7 Restart VMS.

8 Navigate to **Admin > Setup > Configuration > modules > cp > repository > path.processor.plugin.classes.**

9 Add 'SamplePathProcessorPlugin' to the existing list. This should be the seam annotation name.



Triggering the Sample Plug-in

10 Create an Action Profile.

- **a** Navigate to **Workflow > Manage > Action Profiles > Add New.**
- **b** Enter a name.
- **c** Select "Repository Manager" from the drop-down menu.
- **d** Click **Save & Edit.**
- **e** Set the command to: Copy.
- **f** Click **Add** to include the destination.
- **g** Set the file path to: File only.
- **h** Set Path Processor to: SamplePathProcesssorPlugin.
- **i** Click **Save** then **Activate.**

11 From an existing workflow definition in a logical video, navigate to the **Configure tab > Publish to CDN > Processor.**

   a Select SamplePathProcessorPlugin.

   b Click **Save.**

12 Place the video/`.jpg`/`.xml` files into the Hot Folder/Logical Video/In folder.

Expected Results

This sample sets the destination folder of the file as a time-stamped folder. The copied files should appear in the <Destination Folder/Time Stamped Folder/file>.

Chapter 6

Creating Custom Entitlement Manager Plug-ins

This chapter includes information about creating the following custom Entitlement Manager plug-ins:

Entitlement Manager

Prerequisites

The following steps are applicable to each of the Entitlement Manager plug-ins listed below.These build and deploy steps must be performed before configuring or triggering the plug-ins.

1. Install the parent POM file into the local Maven repository.
2. Generate the plug-in archetype.
3. Create a "Project Root" folder on a local computer to house the custom plug-in(s).
4. Build the plug-in files and place in the "Project Root" folder.

```
cd <Project root>
mvn archetype:generate
-DarchetypeGroupId=com.xxx.vms.plugins
-DarchetypeArtifactId=vms-em-plugins-archetype
-DarchetypeVersion=x.x-xxxxxx (plugins version)
-DgroupId=com.xxx.vms
-DartifactId=custom-cm-plugin
```

```
-Dversion=1.25
```

5 Build a signed or unsigned JAR file.

6 Deploy the JAR file into the VMS server.

a Navigate to **Configure > Custom File Deployment > Add New.**

b Enter a name.

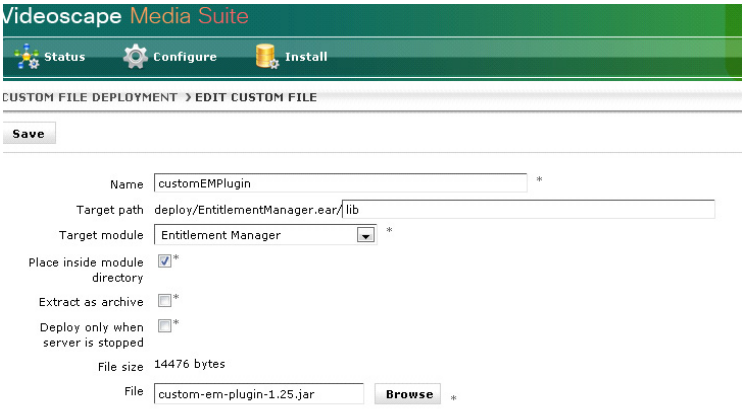
c Set the target path to: deploy/EntitlementManager.ear/lib.

d Set the target module to: Entitlement Manager.

e Check the box “Place inside module directory”.

f Browse and select the created JAR file.

g Click **Save** then **Activate.**



7 Restart VMS.

Account Manager Plug-in

Plug-in Architecture

To view the plug-in architecture, see the attached Plug-in Architecture zip file.

Description

Custom authentication and customer account management functions are implemented using the `com.extend.opencase.em.pluggable.account.AccountManagerPlugin` interface. A default `AccountManagerPlugin` implementation is defined at a system level, and different implementations can be defined per affiliate. This plug-in provides custom authentication, CRUD pre-operation, CRUD post-operation, encryption and decryption of passwords, and reset password functionality for household accounts.

To view the methods this interfaces specifies, open the Entitlement Manager folder in “vms_docs-[releasenumber]-[buildnumber]” and select “index.html”.

Primary Use Case:

This plug-in is used to perform authentication checks on a third party application and then notify the third party application about CRUD operations.

Example

The following is a sample Account Manager plug-in.

```
public class SampleAccountManagerPlugin implements AccountManagerPlugin {

	private static final transient Log LOG = LogFactory.getLog(SampleAccountManagerPlugin.class);

	@Override
	public Login authenticate(Credentials credentials) throws AuthenticationException, AccountManagerPluginException
{
		LOG.info("Authenticate with SampleAccountManagerPlugin with credentials: " + credentials.toString());
		return null;
	}

	@Override
	public String decryptPassword(Login login) throws AccountManagerPluginException {
		LOG.info("DecryptPassword for password [" + login.getPassword() + "]");
		return null;
	}

	@Override
	public String encryptPassword(Login login) throws AccountManagerPluginException {
		LOG.info("EncryptPassword for password [" + login.getPassword() + "]");
		return null;
	}

	@Override
	public String getOAuthSecret(Login login) throws AccountManagerPluginException {
		LOG.info("geOAuthSecrete for login with email [" + login.getEmail() + "] and password ["+
		login.getPassword() + "]");
		return null;
	}

	@Override
	public void initialize(String initData) throws PluginInitializationException {
		if (null != initData && !initData.isEmpty()) {
			LOG.info("SampleAccountManagerPlugin initialized with value [" + initData + "].");
			return;
		}
		LOG.info("SampleAccountManagerPlugin initialized with no configuration.");
	}

	@Override
	public boolean isPasswordResettable() {
		LOG.info("Password reset is disabled.");
		return false;
	}

	@Override
	public void preCreate(Account account) throws AccountManagerPluginException {
		LOG.info("Account with email [" + account.getEmail() + "] will be created.");
	}

	@Override
	public void postCreate(Account account) throws AccountManagerPluginException {
		LOG.info("Account with email [" + account.getEmail() + "] is created.");
	}

	@Override
	public void preUpdate(Account account) throws AccountManagerPluginException {
		LOG.info("Account with email [" + account.getEmail() + "] will be updated.");
	}

	@Override
	public void postUpdate(Account account) throws AccountManagerPluginException {
		LOG.info("Account with email [" + account.getEmail() + "] is updated.");
```

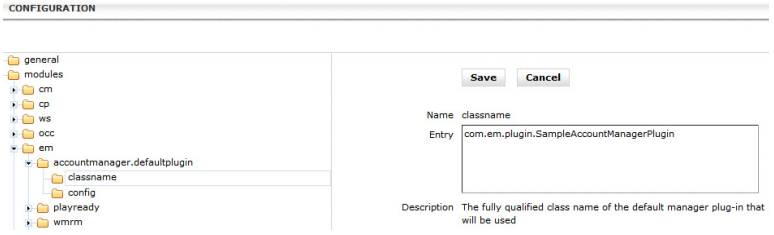
```
        }

        @Override
        public void preDelete(Account account) throws AccountManagerPluginException {
                LOG.info("Account with email [" + account.getEmail() + "] will be deleted.");
        }

        @Override
        public void postDelete(Account account) throws AccountManagerPluginException {
                LOG.info("Account with email [" + account.getEmail() + "] is deleted.");
        }

        @Override
        public boolean resetPassword(Login login, NameValuePairs pairs) throws AccountManagerPluginException {
                LOG.info("Password reset disabled.");
                return false;
        }
}
```

Configuring the Plug-in

Perform the following steps to configure a custom Account Manager plug-in:

1 Add a new Account Manager plug-in.

- **a** Navigate to **Admin > Setup > Configuration > modules > em > accountmanager.defaultplugin > classname.**
- **b** Enter the qualified class name of the custom plug-in in the Entry field. For example: com.em.plugin.SampleAccountManagerPlugin
- **c** Click **Save**.



Triggering the Sample Plug-in

2 Create a new account.

- **a** Navigate to **Entitlement > Accounts > Household > Add New.**
- **b** Enter a value in the mandatory fields.
- **c** Click **Save** then **Activate.**

Expected Results

This sample plug-in tracks the CRUD operation of an account, disable password reset, and authentications.

DRM Transaction Manager Plug-in

Plug-in Architecture

To view the plug-in architecture, see the attached Plug-in Architecture zip file.

Description

The DRM Transaction Manager plug-in allows integrators to add custom functionality to Media Suite's DRM request lifecycle. This plug-in provides custom CRUD pre- and post-operation and pre- and post-status updates of a DRM transaction.

To view the methods this interfaces specifies, open the Entitlement Manager folder in "vms_docs-[releasenumber]-[buildnumber]" and select "index.html".

Primary Use Case:

The DRM Transaction Manager Plug-in has many uses. For example, the postCommit handler can be used when an entitlement license is issued for the first time. When this happens, the plug-in sends a notification to an external billing system.

Example

The following is a sample DRM Transaction Manager plug-in.

```
public class SampleDrmTxManagerPlugin implements DrmTxManagerPlugin {

          private static final transient Log LOG = LogFactory.getLog(SampleDrmTxManagerPlugin.class);

          @Override
          public void initialize(String initData) throws PluginInitializationException {
                    if (null != initData && !initData.isEmpty()) {
                              LOG.info("SampleDrmTxManagerPlugin initialized with value [" + initData + "].");
                              return;
                    }
                    LOG.info("SampleDrmTxManagerPlugin has no initial configuration.");
          }

          @Override
          public void preCreate(DrmTx drmTx) throws DrmTxManagerPluginException {
                    LOG.info("SampleDrmTxManagerPlugin, DRM Transaction will be created with uuid [" + drmTx.getUuid() +
                    "].");
          }

          @Override
          public void postCreate(DrmTx drmTx) throws DrmTxManagerPluginException {
                    //Notify 3rd party that a DRM transaction is started
                    LOG.info("SampleDrmTxManagerPlugin, DRM Transaction is created with uuid [" + drmTx.getUuid() + "].");
          }

          @Override
          public void preCommit(DrmTx drmTx) throws DrmTxManagerPluginException {
                    LOG.info("SampleDrmTxManagerPlugin, DRM transaction with uuid [" + drmTx.getUuid() + "] will be
                    committed.");
          }

          @Override
          public void postCommit(DrmTx drmTx) throws DrmTxManagerPluginException {
                    LOG.info("SampleDrmTxManagerPlugin, DRM transaction with uuid [" + drmTx.getUuid() + "] is committed.");
          }

          @Override
```

```
        public void preStart(DrmTx drmTx) throws DrmTxManagerPluginException {
                LOG.info("SampleDrmTxManagerPlugin, DRM transaction will be started for uuid [" + drmTx.getUuid() +
                "].");
        }

        @Override
        public void postStart(DrmTx drmTx) throws DrmTxManagerPluginException {
                LOG.info("SampleDrmTxManagerPlugin, DRM transaction is started for uuid [" + drmTx.getUuid() + "].");
        }

        @Override
        public void preUpdate(DrmTx drmTx) throws DrmTxManagerPluginException {
                LOG.info("SampleDrmTxManagerPlugin, DRM transaction with uuid [" + drmTx.getUuid() + "] will be
                updated.");
        }

        @Override
        public void postUpdate(DrmTx drmTx) throws DrmTxManagerPluginException {
                LOG.info("SampleDrmTxManagerPlugin, DRM transaction with uuid [" + drmTx.getUuid() + "] is updated.");
        }

        @Override
        public void preUpdateStatus(DrmTx drmTx) throws DrmTxManagerPluginException {
                LOG.info("SampleDrmTxManagerPlugin, DRM Trasaction's status will be updated.");
        }

        @Override
        public void postUpdateStatus(DrmTx drmTx) throws DrmTxManagerPluginException {
                LOG.info("SampleDrmTxManagerPlugin, DRM Trasaction's status is updated.");
        }

        @Override
        public void preDelete(DrmTx drmTx) throws DrmTxManagerPluginException {
                //Notify 3rd party system that the DRM is being deleted.
                LOG.info("SampleDrmTxManagerPlugin, DRM Trasaction will be deleted with UUID [" + drmTx.getUuid() +
                "].");
        }

        @Override
        public void postDelete(DrmTx drmTx) throws DrmTxManagerPluginException {
                //Notify 3rd party system that the DRM is deleted.
                LOG.info("SampleDrmTxManagerPlugin, DRM Trasaction with UUID [" + drmTx.getUuid() + "] is deleted.");
        }
}
```

Configuring the Plug-in

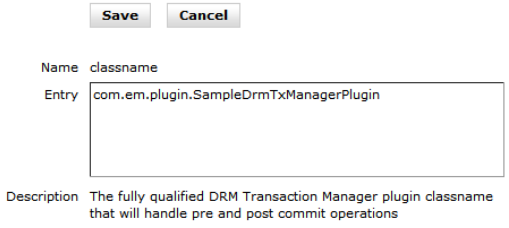
Perform the following steps to configure a custom DRM Transaction Manager plug-in:

1 Add a new DRM Transaction Manager plug-in.

   a Navigate to **Admin > Setup > Configuration > modules > em > drmtxmanager.plugin > classname.**

   b Enter the qualified class name of the custom plug-in in the Entry field. For example: com.em.plugin.SampleDrmTxManagerPlugin.

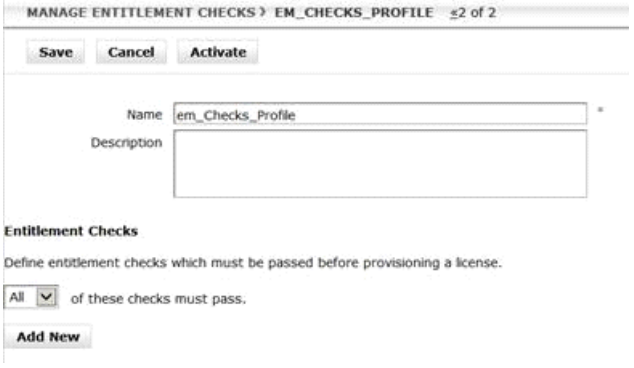
   c Click **Save.**



Triggering the Sample Plug-in

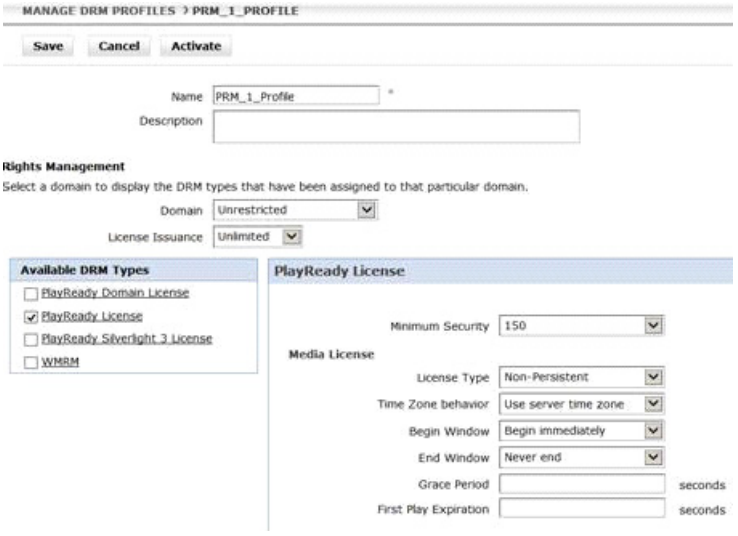
2 Create a new entitlement check profile.

- **a** Navigate to **Entitlement > Productization > Profiles > Entitlement Checks > Add New.**
- **b** Enter a name and description.
- **c** Click **Create & Edit.**
- **d** Click **Save** then **Activate**.



3 Create a DRM profile.

- **a** Navigate to **Entitlement > Productization > Profiles > DRM Profiles > Add New.**
- **b** Enter a name and description.
- **c** Select the PlayReady License check box.
- **d** Click on the PlayReady License link.
- **e** Select the minimum security (any available value).
- **f** Select “Non-Persistent” from the **License Type** drop-down menu.
- **g** Select “Use server time zone” from the **Time Zone behaviour** drop-down menu.
- **h** Select “Begin immediately” from the **Begin Window** drop-down menu.
- **i** Select “Never” from the **End Window** drop-down menu.
- **j** Click **Save** then **Activate**.



4 Create a new policy.

a Navigate to **Entitlement > Productization > Policies > Add New.**

b Enter a name and description.

c Click **Create & Edit.**

d Select the new entitlement check profile from the **Entitlement Check** drop-down menu.

e Select the new DRM profile from the **DRM Profile** drop-down menu.

f Click **Save** then **Activate**.

5 Create a new bundle.

a Navigate to **Metadata > Bundles > Add New > Logical Video.**

b Enter a name and description.

c Click **Create & Edit.**

d Navigate to **Videos > Add New > Physical Asset Video/Manifest.**

e Add a value in the **Content ID** field.

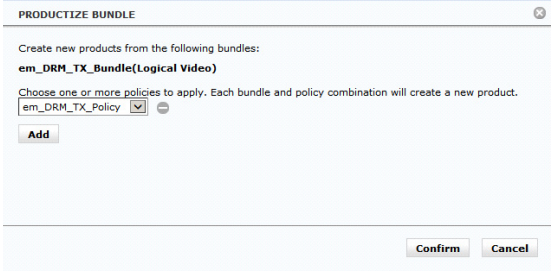
f Click **Save** then **Activate.**

g Click **Productize**.

h Click **Add**.

i Select the new policy (with an entitlement check profile).

j Click **Confirm**.



6 Create a new account.

a Navigate to **Entitlement > Accounts > Household > Add New.**

b Enter data in the mandatory fields.

c Click **Save.**

7 Create an entitlement using the web method `RightsLockerWebService.createEntitlementByAccountIdAndProductId(<AccoundUUIDOfTheAccountCreatedAbove>,<ProductUUIDOfTheProductCreatedAbove>)`

8 Create encryption data in the physical asset video/manifest.

9 Execute the method `PlayReadyProviderService.GetEncryptInfo(inputXml)`.

10 Retrieve the keyID from the **License Request Data** of the physical asset.

11 Retrieve the license request for the new product by using the keyID, Physical Asset UUID, and Entitlement UUID.

Expected Results

Check the log and ensure that the CRUD operation of the DRM transaction (create/update/delete/commit) is logged.

Entitlement Manager Plug-in

Plug-in Architecture

To view the plug-in architecture, see the attached Plug-in Architecture zip file.

Description

The Entitlement Manager plug-in provides notifications about Entitlement lifecycle events to external systems. It provides custom CRUD pre- and post-operation for the life-cycle of the entitlement.

To view the methods this interfaces specifies, open the Entitlement Manager folder in “vms_docs-[releasenumber]-[buildnumber]” and select “index.html”.

Primary Use Case:

This plug-in is vital to use cases where a record of entitlements created in Media Suite must be maintained in an application-neutral data store, such as the DECE "Rights Locker" concept. An example of the type of initialization data that could be passed to an EntitlementManagerPlugin implementation is connection details of an external system to be notified of Media Suite entitlement lifecycle events.

Example

The following is a sample Entitlement Manager plug-in. It tracks the CRUD operations of entitlements, it takes the initial data from the configuration as the productId. If an entitlement is created for that product, the status is set to complete.

```
public class SampleEntitlementManagerPlugin implements EntitlementManagerPlugin {

        private static final transient Log LOG = LogFactory.getLog(SampleEntitlementManagerPlugin.class);

        protected String productId = new String();

        EntitlementWebService entitlementWebservice = EntitlementWebServiceProvider.getEntitlementWebService();

        /**
         * Initialize data for the this instance of plugin. In this case, the value store in the config is expected to be
         product uuid of the entitlement that will be created
         *
         * @param initData The data from database configuration entitlementmanager.plugin -> config
         */
        @Override
        public void initialize(String initData) throws PluginInitializationException {
                if (null != initData && !initData.isEmpty()) {
                        LOG.info("Initial data found with data [" + initData + "]");
                        productId = initData;

                        try {
                                Product product = entitlementWebservice.findProductByUuid(productId);
                                if (null == product) {
                                        LOG.info("Unable to find product. with id [" + productId + "].");
                                        productId = null;
                                }
                        } catch (EntitlementParameterFault e) {
                                LOG.info("Unable to find product. ", e);
                                productId = null;
                        } catch (EntitlementWebserviceFault e) {
                                LOG.info("Unable to find product. ", e);
                                productId = null;
                        }

                } else {
                        LOG.info("No productId initData, no action will be performed on the entitlement");
                }

        }
```

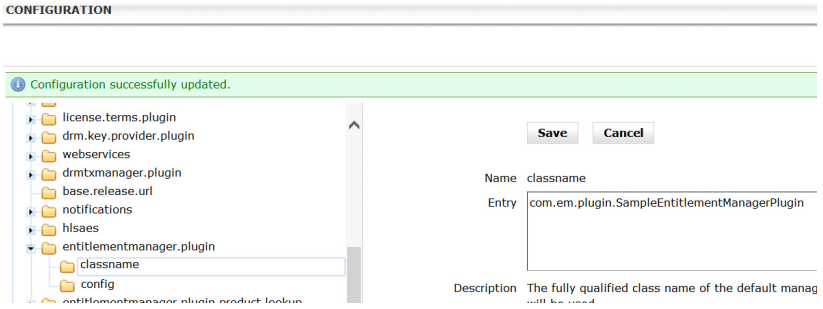
```
        /**
         * This method is run before creating an entitlement
         *
         * @param entitlement Entitlement that will be created
         */
        @Override
        public void preCreate(Entitlement entitlement) throws EntitlementManagerPluginException {
                 LOG.info("Entitlement will be created with accountId [" + entitlement.getAccount().getUuid() + "] and
                 productId [" + entitlement.getProductId() + "].");
                 if (null != productId && entitlement.getProductId().equals(productId)) {
                          Date currentTime = new Date(50000);
                          entitlement.setCreateDate(currentTime);
                          LOG.info("Create entitlement with create date " + currentTime.toString());
                 }
        }

        /**
         * This method is run after creating an entitlement
         *
         * @param entitlement Entitlement that is created
         */
        @Override
        public void postCreate(Entitlement entitlement) throws EntitlementManagerPluginException {
                 LOG.info("Entitlement created with accountId [" + entitlement.getAccount().getUuid() + "] and productId
                 [" + entitlement.getProductId() + "].");
                 if (null != productId && entitlement.getProductId().equals(productId)) {
                          entitlement.setStatus(EntitlementStatus.COMPLETE);
                          LOG.info("Entitlment with uuid [" + entitlement.getUuid() + "] has status set to completed.");
                 }
        }

        /**
         * This method is run before updating an entitlement
         *
         * @param entitlement Entitlement that will be updated
         */
        @Override
        public void preUpdate(Entitlement entitlement) throws EntitlementManagerPluginException {
                 LOG.info("Entitlement with accountId [" + entitlement.getAccount().getUuid() + "] will be updated.");
        }

        /**
         * This method is run after updating the entitlement
         *
         * @param entitlement Entitlement that is updated
         */
        @Override
        public void postUpdate(Entitlement entitlement) throws EntitlementManagerPluginException {
                 LOG.info("Entitlement with accountId [" + entitlement.getAccount().getUuid() + "] updated.");
        }

        /**
         * This method is run before deleting an entitlement
         *
         * @param entitlement Entitlement that will be deleted
         */
        @Override
        public void preDelete(Entitlement entitlement) throws EntitlementManagerPluginException {
                 LOG.info("Entitlement with uuid [" + entitlement.getUuid() + "] will be deleted.");
        }

        /**
         * This method is run after an entitlement is deleted
         *
         * @param entitlement Entitlement that is deleted
         */
        @Override
        public void postDelete(Entitlement entitlement) throws EntitlementManagerPluginException {
                 LOG.info("Entitlement with uuid [" + entitlement.getUuid() + "] is deleted.");
        }
}
```

Configuring the Plug-in

Perform the following steps to configure a custom Entitlement Manager plug-in:

1 Add a new Entitlement Manager plug-in.

- **a** Navigate to **Admin > Setup > Configuration > modules > em > entitlementmanager.plugin > classname**.
- **b** Enter the qualified class name of the custom plug-in in the Entry field.For example: com.em.plugin.SampleEntitlementManagerPlugin.
- **c** Click **Save**.



Triggering the Sample Plug-in

2 Create a new policy.

- **a** Navigate to **Entitlement > Productization > Policies > Add New.**
- **b** Enter a name and description.
- **c** Click **Create & Edit**.
- **d** Click **Save** then **Activate.**

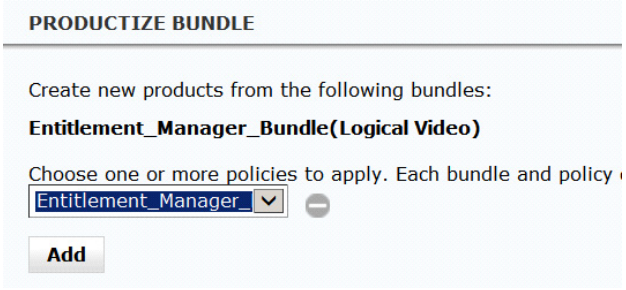
3 Create a new bundle.

- **a** Navigate to **Metadata > Bundles > Add New > Logical Video.**
- **b** Enter a name and description.
- **c** Click **Create & Edit.**
- **d** Add desired components using the **Add New** button within each folder.
- **e** Add data into the desired fields for each component.
- **f** Click **Save** then **Activate.**

4 Click **Productize**.

5 Click **Add.**

6 Select the policy then click **Confirm.**



7 Navigate to **Admin > Setup > Configuration > modules > em > entitlementmanager.plugin > config**

   a Enter the productUUID.

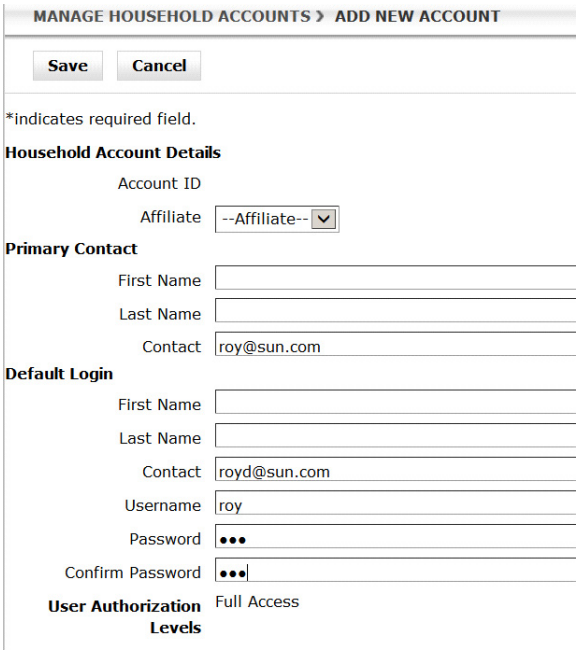
   b Click **Save**.

8 Create a new account.

   a Navigate to **Entitlement > Accounts > Household > Add New.**

   b Add data in the desired fields.

   c Click **Save** then **Activate.**



9 Create an entitlement for the product using the following web service method:
`RightsLockerWebService.createEntitlementByAccountIdAndProductId(<AccoundUUIDOfTheAccountCreatedAbove>,<ProductUUIDOfTheProductCreatedAbove>)`

10 Update the entitlement using `RightsLockerWebService.createEntitlement`

11 Create a new product.

12 Create an entitlement for the new product using the web method

RightsLockerWebService.createEntitlementByAccountIdAndProductId(<AccoundUUIDOfTheAccountCreatedAbove>,<ProductUUIDOfTheProductCreatedAbove>)

Expected Results

CRUD operations of entitlements are logged in the server log. Entitlement create actions with the productId specified in the configuration have a status set to complete.

Entitlement Check Plug-in

Plug-in Architecture

To view the plug-in architecture, see the attached Plug-in Architecture zip file.

Description

Custom initialization data can be passed to EntitlementCheckPlugin instances on instantiation by implementing the `initialize(NameValuePairs)` method. The initialize method takes a collection of name/value pair strings that can be configured on the Media Suite user interface. The Entitlement Check plug-in checks whether a user has the required rights to watch a video.

To view the methods this interfaces specifies, open the Entitlement Manager folder in "vms_docs-[releasenumber]-[buildnumber]" and select "index.html".

Primary Use Case:

The primary use case for this plug-in is to perform entitlement checks based on entitlement status, account status, and products related to a specific account. It can also be used to perform an entitlement check against another system.

Example

The following is a sample Entitlement Check plug-in. It will perform an entitlement check based on the initial value of the entitlement profile.

```
public class SampleEntiltmentCheckPlugin implements EntitlementCheckPlugin {

        private static final transient Log LOG = LogFactory.getLog(SampleEntiltmentCheckPlugin.class);

        protected static final String PRODUCT_ID = "productId";

        protected String productId = new String();

        /**
         * Checks entitlement based on the specified credentials.
         *
         * @param account
         * @param credentials
         *            A Credentials object containing the necessary identifiers to complete the entitlement check.
         * @param product
         * @param drmTx
         * @param entitlementCheckResourceType
         *            indicate what type of request is being processed
         * @throws EntitlementCheckException
         */
        @Override
        public void checkEntitlement(Account account, Product incomingProduct, Credentials credentials, DrmTx drm,
```

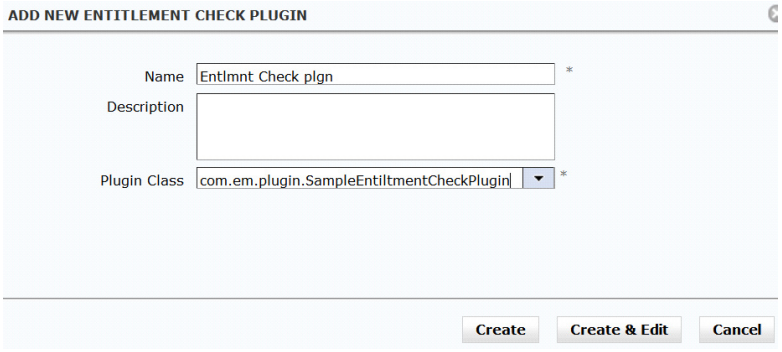
```
                        EntitlementCheckResourceType checkResourceType) throws EntitlementCheckException {

                EntitlementWebService webservice = EntitlementWebServiceProvider.getEntitlementWebService();
                if (productId != null && productId.trim() != "") {
                        LOG.info("SampleEntiltmentCheckPlugin invoked with productId [" + productId + "], and incoming
                        UUID [" + incomingProduct.getUuid()
                                        + "].");
                        try {
                                Product product = webservice.findProductByUuid(productId);
                                if (null == product) {
                                        LOG.info("Entitlement check failed, unable to find product with uuid [" +
                                        productId + "].");
                                        throw new EntitlementCheckException("Entitlement check failed, unable to
                                        find product with uuid [" + productId + "].");
                                }
                                if (product.getUuid().equalsIgnoreCase(incomingProduct.getUuid())) {
                                        LOG.info("Incoming product and input product share the same UUID,
                                        entitlmeent check pass.");
                                } else {
                                        LOG.info("Entitlement check failed. Incoming product and input product are
                                        different. Find product name ["
                                                        + product.getName() + "], incoming product name [" +
                                                        incomingProduct.getName() + "].");
                                        throw new EntitlementCheckException(
                                                        "Entitlement check failed. Incoming product and input
                                                        product are different. Find product name ["
                                                                        + product.getName() + "], incoming
product name [" + incomingProduct.getName() + "].");
                                }

                        } catch (EntitlementParameterFault e) {
                                LOG.error("Unable to find product. ", e);
                        } catch (EntitlementWebserviceFault e) {
                                LOG.error("Unable to find product. ", e);
                        }
                } else {
                        LOG.info("Entitlement check failed, make sure productId field is set in the plugin
                        configuration.");
                        throw new EntitlementCheckException("Entitlement check failed, make sure productId field is
                        set in the plugin configuration.");
                }
        }

        /**
         * Initialize the productId of the allowed product
         *
         * @param nameParisHash map of value between Name and Values, they are input when an entitlement profile is
         created
         * @throws PluginInitializationException
         */
        @Override
        public void initialize(NameValuePairs nameParis) throws PluginInitializationException {
                try {
                        for (NameValuePair pair : nameParis.getNameValuePair()) {
                                if (pair.getName().equals(PRODUCT_ID)) {
                                        productId = pair.getValue();
                                } else {
                                        productId = new String();
                                }
                        }
                } catch (Exception e) {
                        throw new PluginInitializationException("Failed to initialize SampleEntiltmentCheckPlugin.",
                        e);
                }
        }
}
```

Configuring the Plug-in

Perform the following steps to configure a custom Entitlement Check plug-in:

1 Add a new Entitlement Check plug-in.

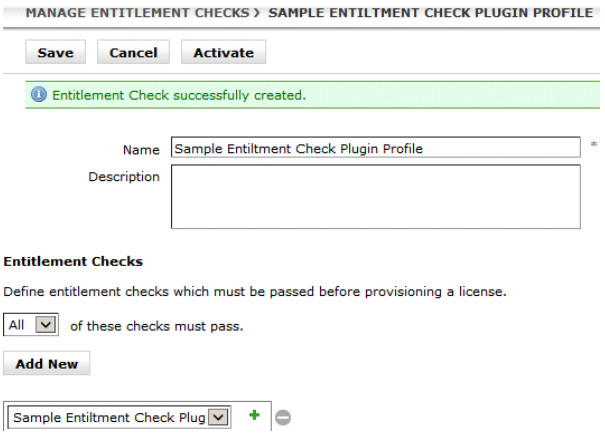
- **a** Navigate to **Entitlement > Setup > Plugins > Entitlement Check Plugins > Add New.**
- **b** Enter a name.
- **c** Set the Plugin Class to: sampleEntitlementCheckPlugin.
- **d** Click **Create & Edit.**
- **e** Click **Save** then **Activate.**



Triggering the Sample Plug-in

2 Create an entitlement check profile for the new plug-in.

- **a** Navigate to **Entitlement > Productization > Profiles > Entitlement Checks > Add New.**
- **b** Enter a name and description.
- **c** Click **Create & Edit.**
- **d** Click **Add New** to define the entitlement checks that must be passed before provisioning a license.
- **e** Select the new entitlement check plug-in from the drop-down menu.
- **f** Click **Save** then **Activate.**



3 Create a new policy.

a Navigate to **Entitlement > Productization > Policies > Add New.**

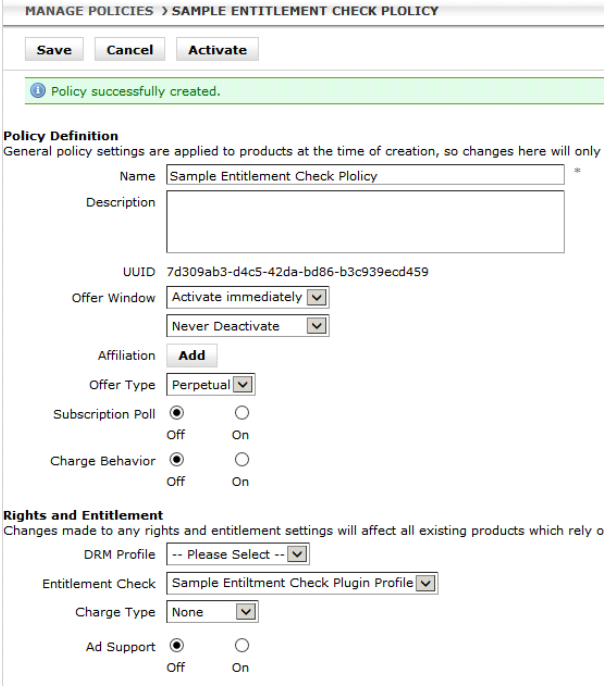
b Enter a name and description.

c Click **Create & Edit.**

d Select the new entitlement check profile from the drop-down menu.

e Enter data in the desired fields.

f Click **Save** then **Activate.**



4 Create a new bundle.

a Navigate to **Metadata > Bundles > Add New > Logical Video**

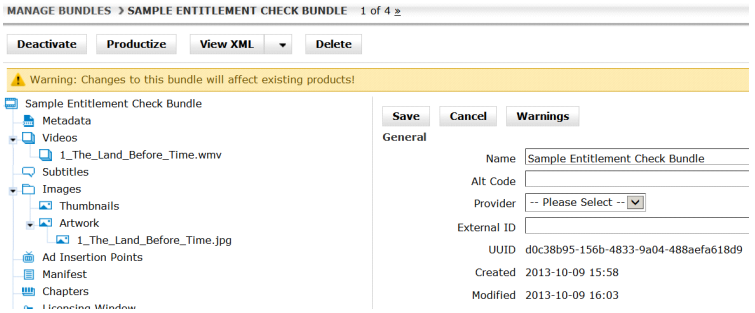
b Enter a name and description.

c Click **Create & Edit.**

d Add desired components using the **Add New** button within each folder.

e Add data into the desired fields for each component.

f Click **Save** then **Activate**.



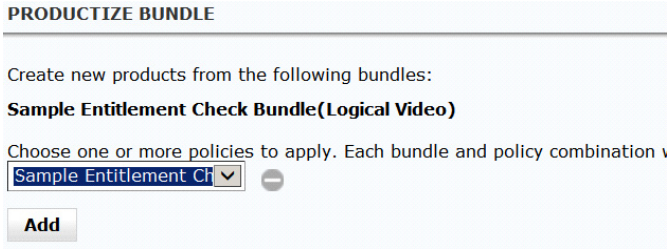
5 Productize the bundle.

a Click the **Productize** button on the **Bundle Details** page.

b Click **Add**.

c Select the new policy (with an entitlement check profile).

d Click **Confirm**.



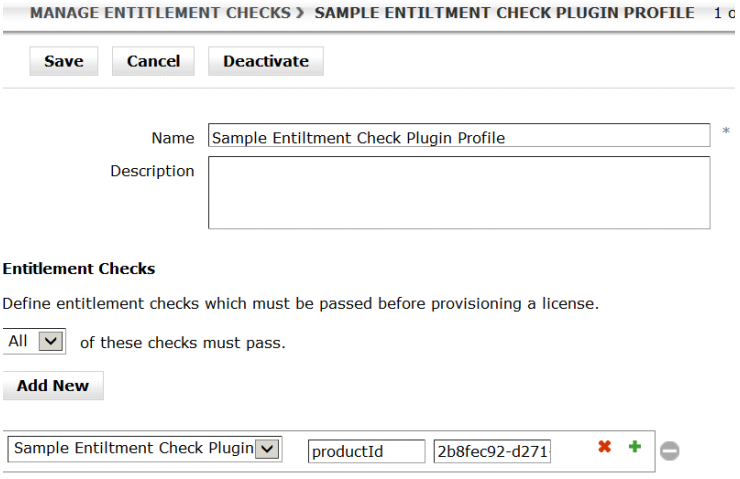
6 Select the entitlement check profile.

a Navigate to **Entitlement > Productization > Profiles > Entitlement Checks.**

b In the text box beside the drop-down menu, enter “productId”.

c In the second text box, enter the product UUID of the new product.

d Click **Save** then **Confirm**.

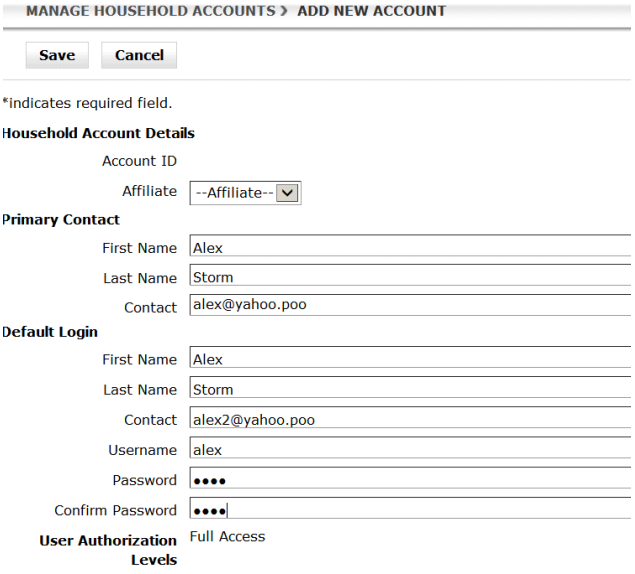


7 Create a new account.

a Navigate to **Entitlement > Accounts > Household > Add New.**

b Enter data in the desired fields.

c Click **Save.**



8 Create an entitlement using the web method

`RightsLockerWebService.createEntitlementByAccountIdAndProductId(<AccoundUUIDOfTheAccountCreatedAbove>,<ProductUUIDOfTheProductCreatedAbove>)`

9 Check the entitlement using the web method

`RightsLockerWebService.checkEntitlement(<loginUUIDOfTheAccountCreatedAbove>,<relevantCredentialsForThePlugin>)`

This check should pass and return the product UUID of the new product.

10 Create another product using the process above.

Note Do not modify the entitlement check profile. It should contain the first product created.

---

11 Create an entitlement for the second product using the web method `RightsLockerWebService.createEntitlementByAccountIdAndProductId(<AccoundUUIDOfTheAccountCreatedAbove>,<ProductUUIDOfTheProductCreatedAbove>)`

12 Check the entitlement for the second product using the web method `RightsLockerWebService.checkEntitlement(<loginUUIDOfTheAccountCreatedAbove>,<credentialsWithOnlyEntitlementIDAndProductID>).`

Expected Results

The parameter is configured in the entitlement profile where productId will be stored. When a check entitlement action occurs, if the productId is configured in the entitlement profile, the entitlement check will pass. Otherwise, it will fail.

Entitlement Manager Notification Plug-in

Plug-in Architecture

To view the plug-in architecture, see the attached Plug-in Architecture zip file.

Description

The Entitlement Manager Notification plug-in sends notifications to a third party application if a change occurs in a login object, entitlement object, or account object. When an event occurs a notification is created for that object.

To view the methods this interfaces specifies, open the Entitlement Manager folder in "vms_docs-[releasenumber]-[buildnumber]" and select "index.html".

Primary Use Case:

This plug-in is used to notify a third party application about events for a specific object.

Example

The following is a sample Entitlement Manager Notification plug-in. It logs the CRUD operation of an entitlement object. If the action is create, additional details will be logged.

```
public class SampleEntitlementNotificationPlugin implements NotificationPlugin {

        private static final transient Log LOG = LogFactory.getLog(SampleEntitlementNotificationPlugin.class);

/**
 * Process Entitlement Manager notification with custom logic
 *
 * @param object the entity which the event occurred.
 * @param objectPrimaryKey The primary key of the object the notification is triggered for.
 * @param action the life cycle event that occurred.
 * @param notificationData Any optional data associated with the type of object the notification is triggered by.
 * @param notificationCreateDate Creation date of notification.
```

```
 * @throws NotificationPluginException If an error occurs when processing a notification. This will indicate to the
          notification job that the notification should not be deleted and retried later.
 */
@Override
public void processNotification(String object, Integer objectPrimaryKey, String action, String notificationData,//
                            Date notificationCreateDate) throws NotificationPluginException {

                    try {
                              final Class<?> clazz = Class.forName(object);
                              final LifecycleActionsEnum actionEnum = LifecycleActionsEnum.valueOf(action);

                              if (clazz.isInstance(new Entitlement())) {

                                        LOG.info("Invoked SampleEntitlementNotificationPlugin with object ["
                                                  + object + "], PK [" + objectPrimaryKey + "], notificationData ["
                                                  + notificationData + "], Action [" + action.toString()
                                                  + "], notificationCreateDate [" +
                                                  notificationCreateDate.toString() + "].");

                                        if (LifecycleActionsEnum.CREATE.equals(actionEnum)) {

                                                  //Find the entitlement that is created through webservice
                                                  Entitlement entitlment =
getRightsLockerWebservice().findEntitlementByPk(objectPrimaryKey);

                                                  LOG.info("Entitlement created with bundle type [" +
entitlment.getBundleType() + "], productUuid [" + entitlment.getProductId()
                                                                      + "], and accountUuid [" +
entitlment.getAccount().getUuid() + "].");
                                        }
                              }
                    } catch(Exception e) {
                              throw new NotificationPluginException("Failed to process Notification", e);
                    }
                    return;
          }
```

Local webservice can be retrieved through the following method

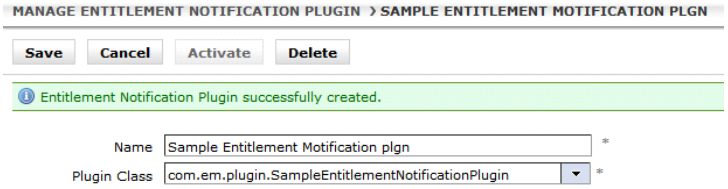
```
          // Retrive the local interface for webservice
          protected RightsLockerWebServiceLocal getRightsLockerWebservice() {
                    return (RightsLockerWebServiceLocal)Component.getInstance(RightsLockerWebService.class);
}
```

Configuring the Plug-in

Perform the following steps to configure a custom Notification plug-in:

1 Add a new Entitlement Manager Notification plug-in.

- **a** Navigate to **Entitlement > Setup > Plugins > Entitlement Notification Plugins > Add New.**
- **b** Enter a name.
- **c** Click **Create & Edit.**
- **d** Set the plugin class to: SampleEntitlementNotificationPlugin.
- **e** Click **Save** then **Activate.**



Triggering the Sample Plug-in

2 Create a new policy.

- **a** Navigate to **Entitlement > Productization > Policies > Add New.**
- **b** Enter a name and description.
- **c** Click **Create & Edit.**
- **d** Click **Save** then **Activate.**

3 Create a new bundle.

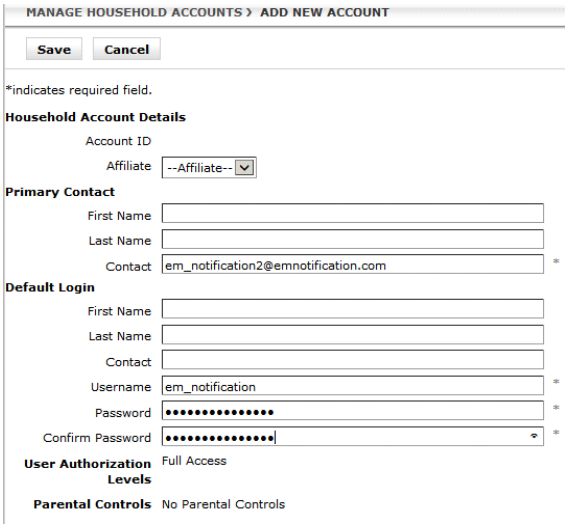
- **a** Navigate to Metadata > Bundles > Add New > **Logical Video**
- **b** Enter a name and description.
- **c** Click **Create & Edit.**
- **d** Click **Add New** to add a video to the bundle.
- **e** Click **Save** then **Activate.**
- **f** Click **Productize.**

4 Click **Add** in the **Productize Bundle** window.

- **a** Select the new policy (with an entitlement check profile).
- **b** Click **Confirm**.

5 Create a new account.

- **a** Navigate to **Entitlement > Accounts > Household > Add New.**
- **b** Click **Save** then **Activate**.



6 Create an entitlement for the product using the web method `RightsLockerWebService.createEntitlementByAccountIdAndProductId(<AccoundUUIDOfTheAccountCreatedAbove>,<ProductUUIDOfTheProductCreatedAbove>)`

7 Update the entitlement using the web method `RightsLockerWebService.updateEntitlement` or delete the entitlement using the web method `RightsLockerWebService.deleteEntitlement`

8 Check the log to ensure that changes to the entitlement are logged.

Expected Results

The CRUD operation of the entitlement object is logged. When creating an entitlement, the bundle type of the product and the account UUID is also returned.

Subscription Product Resolution Plug-in

Plug-in Architecture

To view the plug-in architecture, see the attached Plug-in Architecture zip file.

Description

This class is responsible for returning a sorted list of products which may entitle the user to access content being requested under the context of a subscription. Within a subscription, products are prioritized and content is checked based on the priority of the listed products.

To view the methods this interfaces specifies, open the Entitlement Manager folder in "vms_docs-[releasenumber]-[buildnumber]" and select "index.html".

Primary Use Case:

This plug-in is used for connecting to a third party system to retrieve a sorted list of products which may entitle a user to access content.

Example

The following is a sample Subscription Product Resolution plug-in. It sorts products by name.

```
public class SampleSubscriptionProductResolutionPlugin implements SubscriptionProductResolutionPlugin {

        private static final transient Log LOG = LogFactory.getLog(SampleSubscriptionProductResolutionPlugin.class);

        EntitlementWebService entitlementWebservice = EntitlementWebServiceProvider.getEntitlementWebService();

        @Override
        public List<Product> getOwnedSubscriptionProduct(List<Product> products, Account account, Credentials
        credentials)
                        throws NotEntitledException, SubscriptionProductResolutionPluginException {

                List<String> productUuids = new ArrayList<String>();
                List<Product> sortedProducts = null;
                //List of product will be sorted by parameter 'name'
                List<String> sortParams = new ArrayList<String>();
                sortParams.add("name");


                //List of product cannot be empty
                if (null == products || products.isEmpty()) {
                        throw new
SubscriptionProductResolutionPluginException(EntitlementManagerErrorConstant.error428, //
                                                "Product list cannot be empty");
                }

                //Pick out a unique list of uuid to sort
                for (Product product : products) {
                        if (!productUuids.contains(product.getUuid())) {
                                productUuids.add(product.getUuid());
                        }
                }

                try {
                        //Sort the product by name through webservice
                        sortedProducts = entitlementWebservice.findProductsByUuidAndSortByCriteria(productUuids,
                        Boolean.TRUE, sortParams);
                        StringBuilder productNamesBuilder = new StringBuilder();
                        productNamesBuilder.append("SampleSubscriptionProductResolutionPlugin, The product are sorted
                        in list [");
                        for (Product sortedProduct : sortedProducts) {
                                productNamesBuilder.append("'" + sortedProduct.getName() +  "' ");
                        }
                        productNamesBuilder.append("]");
                        LOG.info(productNamesBuilder.toString());
                } catch (EntitlementParameterFault e) {
                        LOG.error("Unable to find sorted list of product by name with count [" + productUuids.size() +
                        "].");
                } catch (EntitlementWebserviceFault e) {
                        LOG.error("Unable to find sorted list of product by name with count [" + productUuids.size() +
                        "].");
                }

                return sortedProducts;
        }
}
```

Configuring the Plug-in

Perform the following steps to configure a custom Subscription Product Resolution plug-in:

1 Add a new Subscription Product Resolution plug-in.

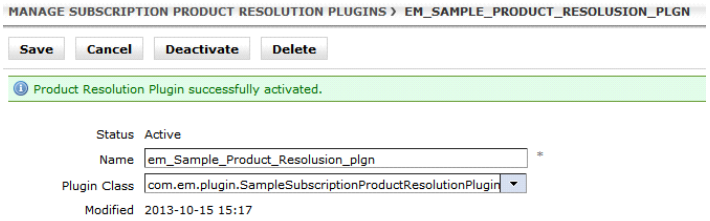
   a Navigate to **Entitlement > Setup > Plugins > Product Resolution Plugins > Add New.**

b Enter a name.

c Click **Create & Edit.**

d Set the plugin class to: SampleSubscriptionProductResolutionPlugin.

e Click **Save** then **Activate.**



Triggering the Sample Plug-in

2 Create a new entitlement check profile.

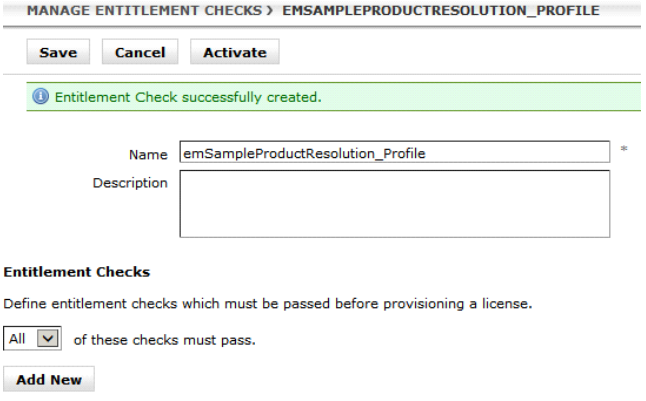
a Navigate to **Entitlement > Productization > Profiles > Entitlement Checks > Add New.**

b Enter a name and description.

c Click **Create & Edit.**

d Click **Save** then **Activate**.

Note Do not select an entitlement check plug-in, as the default entitlement check should be used.



3 Create a new policy.

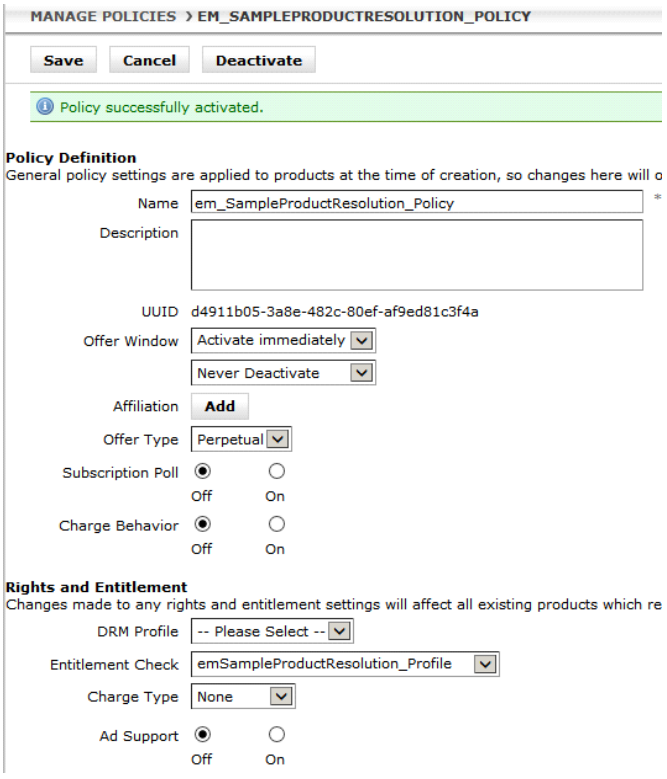
a Navigate to **Entitlement > Productization > Policies > Add New.**

b Enter a name and description.

c Click **Create & Edit.**

d Select the new entitlement check from the drop-down menu.

e Click **Save** then **Activate.**



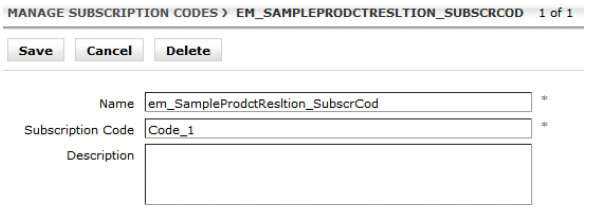
4 Create a new subscription code(s).

a Navigate to **Entitlement > Productization > Subscriptions > Subscription Codes > Add New.**

b Enter a name.

c Set a value in the **Subscription Code** field.

d Click **Create & Edit**.



5 Create a new subscription bundle.

a Navigate to **Metadata > Bundles > Add New > Subscription.**

b Enter a name and description.

c Click **Create & Edit.**

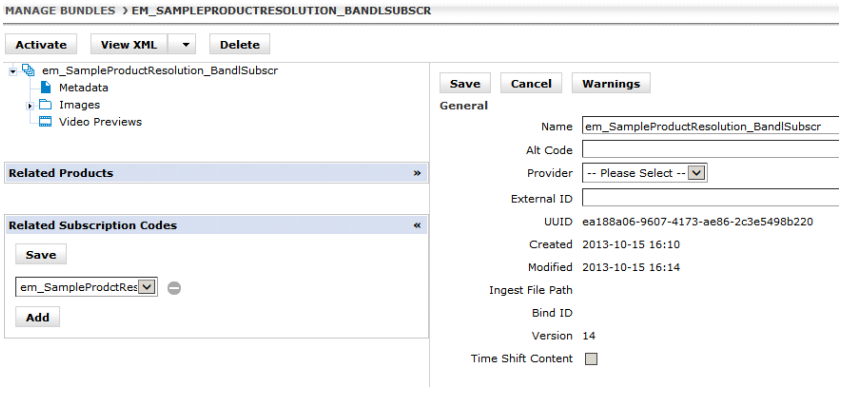
d Click **Save**.

e Click on **Related Subscription Codes**.

f Click **Add** and select the new subscription code(s) from the drop-down menu.

g Click **Save.**

h Click **Activate** then **Productize.**



6 Click **Add** in the **Productize Bundle** window.

a Select the new policy (with an entitlement check profile).

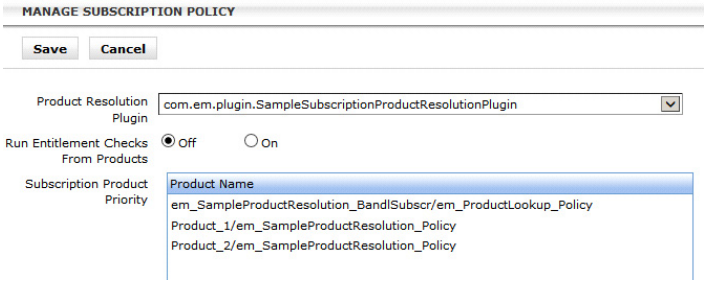
b Click **Confirm**.

7 Create 2 additional bundles following steps 6-7.

8 Navigate to **Entitlement > Productization > Subscriptions > Subscription Policy.**

a Click **Save.**

9 Select the custom Product Resolution plug-in from the **Product Resolution** drop-down menu.



10 Create a new account.

a Navigate to **Entitlement > Accounts > Household > Add New.**

b Click **Save** then **Activate.**

11 Create an entitlement using the web method `RightsLockerWebService.createEntitlementByAccountIdAndProductId(<AccoundUUIDOfTheAccountCreatedAbove>,<ProductUUIDOfTheSubscriptionProductCreatedAbove>)`

12 Check the entitlement using the web method `RightsLockerWebService.checkEntitlement(<loginUUIDOfTheAccountCreatedAbove>,<credentialsWithOnlySubscriptionBundleID>).`

Expected Results

The log will display a list of products that are sorted by name.

Account Validation Plug-in

Plug-in Architecture

To view the plug-in architecture, see the attached Plug-in Architecture zip file.

Description

A description of this plug-in will be provided ASAP.

To view the methods this interfaces specifies, open the Entitlement Manager folder in “vms_docs-[releasenumber]-[buildnumber]” and select “index.html”.

Example

```
package com.em.plugin;

import org.jboss.seam.Component;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.extend.opencase.em.model.account.Account;
import com.extend.opencase.em.pluggable.validation.EntitlementValidationPlugin;
import com.extend.opencase.em.pluggable.validation.exception.EntitlementValidationPluginException;
import com.extend.opencase.em.webservice.AccountManagerWebService;
import com.extend.opencase.em.webservice.AccountManagerWebServiceLocal;
import com.extend.opencase.em.webservice.fault.AccountManagerWebServiceFault;

/**
* SampleAccountValidationPlugin validates Account prior to creating or updating.
*/
public class SampleAccountValidationPlugin implements EntitlementValidationPlugin<Account> {
               private static final transient Logger LOG = LoggerFactory.getLogger(SampleAccountValidationPlugin.class);
               private static final int MAX_LENGTH = 255;

               /**
               * Constructor
               */
               public SampleAccountValidationPlugin() {
                              super();
               }

               /**
               * Validate account creation before preset. This sample plugin checks that the email is no longer than the
maximum character
               * and no primary key is given prior to creation. It also checks that the account has a default login.
               * @param object Account object that is to be created
               * @throws EntitlementValidationPluginException if validation fails Account will not be created
               */
               @Override
               public void validateCreate(Account object) throws EntitlementValidationPluginException {
                              LOG.info("SampleAccountValidationPlug validate Create invoked.");
                              //validate that account primary key is not preset
                              if(object.getAccountPk() != null) {
                                             LOG.debug("Cannot create account with preset primary key.");

                                             try {
                                                            AccountManagerWebServiceLocal accountWebService = getAccount-
ManagerWebService();
                                                            Account existingAccount = accountWebService.findAccount-
ByPk(object.getAccountPk());
                                                            if(existingAccount != null) {
                                                                           LOG.debug("Account already exists [" + existin-
gAccount.getAccountPk() + "].");
```

```
                                                                                throw new EntitlementValidationPluginExcep-
tion("Account already exists [" + existingAccount.getAccountPk() + "].");
                                                                        }
                                                } catch (AccountManagerWebServiceFault e) {
                                                                LOG.debug("Cannot find exisiting account", e);
                                                }
                                                throw new EntitlementValidationPluginException("Cannot create account with
preset primary key.");
                                }

                                //validate that the account email is less than the maximum characters allowed
                                if(!object.getEmail().isEmpty() && object.getEmail().length() > MAX_LENGTH) {
                                                LOG.debug("Account email ["+ object.getEmail() +"] is longer than maximum num-
ber of characters [" + MAX_LENGTH + "] allowed.");
                                                throw new EntitlementValidationPluginException("Account email is longer than
the maximum number of characters allowed.");
                                }

                }

                /**
                * Validate account update. Sample plugin validates that the account exists.
                * Also validates that UUID is not modified and maximum email length. Lastly it checks if
                * object has default login.
                *@param object Account to be updated
                *@throws EntitlementValidationPluginException if validation fails.
                */
                @Override
                public void validateUpdate(Account object) throws EntitlementValidationPluginException {
                                LOG.info("SampleAccountValidationPlug validate Update invoked.");
                                //validate that the account exists arleady
                                if(object.getAccountPk() == null) {
                                                LOG.debug("Cannot update account with no primary key");
                                                throw new EntitlementValidationPluginException("Cannot update account with no
primary key");
                                }

                                //validate that the account exists and that the UUID has not changed - generated at persist
                                try {
                                                AccountManagerWebServiceLocal accountWebService = getAccountManagerWebSer-
vice();
                                                Account existingAccount = accountWebService.findAccountByPk(object.getAc-
countPk());
                                                if(existingAccount == null) {
                                                                LOG.debug("Account does not exists.");
                                                                throw new EntitlementValidationPluginException("Account does
not exists.");
                                                }
                                                if(!existingAccount.getUuid().equalsIgnoreCase(object.getUuid())) {
                                                                LOG.debug("UUID modified, cannot update account.");
                                                                throw new EntitlementValidationPluginException("UUID modi-
fied, cannot update account.");
                                                }
                                } catch (AccountManagerWebServiceFault e) {
                                                LOG.debug("Cannot find exisiting account", e);
                                                throw new EntitlementValidationPluginException("Cannot find Existing Account",
e);
                                }
                                //validate that the account email is less than the maximum characters allowed
                                if(!object.getEmail().isEmpty() && object.getEmail().length() > MAX_LENGTH) {
                                                LOG.debug("Account email ["+ object.getEmail() +"] is longer than maximum num-
ber of characters [" + MAX_LENGTH + "] allowed.");
                                                throw new EntitlementValidationPluginException("Account email is longer than
the maximum number of characters allowed.");
                                }

                                //validate account has a default login
                                if(object.getDefaultLogin() == null) {
                                                LOG.debug("Account default login is NULL, cannot update Account.");
                                                throw new EntitlementValidationPluginException("Account default login is NULL,
cannot update Account.");
                                }
```

```
        }

        /**
        *  Retrieve the local interface for webservice
        * @return AccountManagerWebServiceLocal
        */
        protected AccountManagerWebServiceLocal getAccountManagerWebService() {
                return (AccountManagerWebServiceLocal) Component.getInstance(AccountManagerWebSer-
vice.class, true);
        }
}
```

Configuring the Plug-in

Perform the following steps to configure a custom Account Validation plug-in:

1. Navigate to **Admin > Config > module > em > account.validation.plugin > classname**.
2. Enter "`com.em.plugin.SampleAccountValidationPlugin`" as the classname.
3. Click **Save.**
4. Navigate to **Entitlement > Accounts > Household.**
5. Click **Add New**.
6. Enter information into the mandatory fields then click **Save.**
7. Check the log for "`SampleAccountValidationPlugin validate Create invoked`".
8. Open the created account to update it.
9. Make all necessary changes.
10. Click **Save.**
11. Check the log for "`SampleAccountValidationPlugin validate Update invoked`".

Login Validation Plug-in

Plug-in Architecture

To view the plug-in architecture, see the attached Plug-in Architecture zip file.

Description

A description of this plug-in will be provided ASAP.

To view the methods this interfaces specifies, open the Entitlement Manager folder in "vms_docs-[releasenumber]-[buildnumber]" and select "index.html".

Example

```
package com.em.plugin;

import org.jboss.seam.Component;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.extend.opencase.em.model.account.Login;
import com.extend.opencase.em.pluggable.validation.EntitlementValidationPlugin;
```

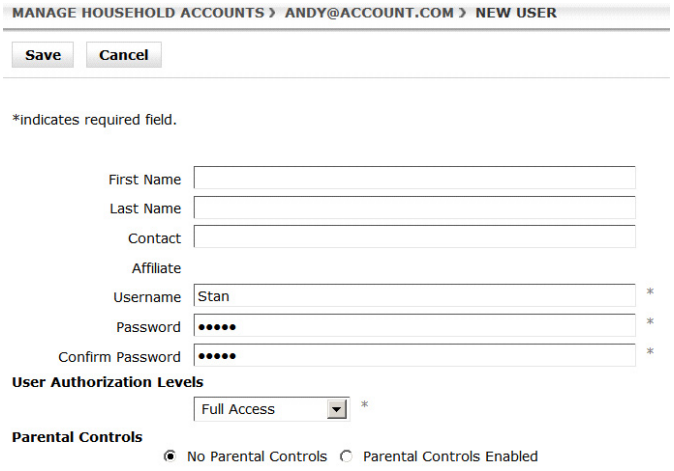
```
import com.extend.opencase.em.pluggable.validation.exception.EntitlementValidationPluginException;
import com.extend.opencase.em.webservice.AccountManagerWebService;
import com.extend.opencase.em.webservice.AccountManagerWebServiceLocal;
import com.extend.opencase.em.webservice.fault.AccountManagerWebServiceFault;



/**
* Sample Login Validation Plugin will be used to validate whenever a login is being created or updated
* through AccountManagerWebService or AffiliateAccountManagerWebService.
* It will also be used to validate a login when it is being activated or deactivated through
 * User Action.
*/
public class SampleLoginValidationPlugin implements EntitlementValidationPlugin<Login> {
               private static final transient Logger LOG = LoggerFactory.getLogger(SampleLoginValidationPlugin.class);
               private static final int MAX_LENGTH = 255;

               /**
               * Validate Create will validate a login object prior to creation of the object. This sample plugin will val-
idate all
                * that this login does not already exist in the database by trying to find a login with
               * the given primary key.
               *
                *  @param object Login object validate
               *  @throws EntitlementValidationPluginException if any validation fails
               */
               @Override
               public void validateCreate(Login object) throws EntitlementValidationPluginException {
                              LOG.info("SampleLoginValidationPlugin invoked. Validating creation of Login.");
                              //login should not be created with primary key - it has to be generated
                              if(object.getLoginPk() != null) {
                                             if(LOG.isDebugEnabled()) {
                                                            LOG.debug("Login cannot be created with preset primary key.");
                                             }

                                             try {
                                                            AccountManagerWebServiceLocal accountWebService = getAccount-
ManagerWebService();
                                                            final Login existingLogin = accountWebService.findLogin-
ByPk(object.getLoginPk());

                                                            if(existingLogin != null) {
                                                                           if(LOG.isDebugEnabled()) {
                                                                                          LOG.debug("Login with primary
key [" + object.getLoginPk() + "] already exists.");
                                                                           }
                                                                           throw new EntitlementValidationPluginExcep-
tion("Login with primary key [" + object.getLoginPk() + "] already exists.");
                                                            }
                                             } catch (AccountManagerWebServiceFault e) {
                                                            if(LOG.isDebugEnabled()) {
                                                                           LOG.debug("Could not find Login with preset pri-
mary key.");
                                                            }
                                             }
                                             throw new EntitlementValidationPluginException("Cannot create login with pre-
set primary key.");
                              }
                              validateTextFields(object);
               }

               /**
               * Validate login prior to updating
               * @param object Login object to validate
               * @throws EntitlementValidationPluginException if validation fails
               */
               @Override
               public void validateUpdate(Login object) throws EntitlementValidationPluginException {
                              LOG.info("SampleLoginValidationPlugin invoked. Validating Login update.");
                              //validate login being updated has a primary key
                              if(object.getLoginPk() == null) {
```

```
                                        if(LOG.isDebugEnabled()) {
                                                LOG.debug("Cannot update Login with no primary key.");
                                        }

                                        throw new EntitlementValidationPluginException("Cannot update Login with no
primary key.");
                                }
                                try {
                                        AccountManagerWebServiceLocal accountWebService = getAccountManagerWebSer-
vice();
                                        final Login existingLogin = accountWebService.findLoginByPk(object.getLog-
inPk());

                                        if(existingLogin == null) {
                                                if(LOG.isDebugEnabled()) {
                                                        LOG.debug("Login does not exist.");
                                                }
                                                throw new EntitlementValidationPluginException("Login does not
exist.");
                                        }
                                } catch (AccountManagerWebServiceFault e) {
                                        if(LOG.isDebugEnabled()) {
                                                LOG.debug("Could not find Login with preset primary key.");
                                        }
                                }
                                validateTextFields(object);

                }

                /**
                * Validate userName, firstName, lastName and Email for maximum length
                * @param object login to be validated
                * @throws EntitlementValidationPluginException if validation fails
                */
                private void validateTextFields(Login object) throws EntitlementValidationPluginException {
                        //validate login text field lengths
                        if(!object.getUsername().isEmpty() && object.getUsername().length() > MAX_LENGTH) {
                                LOG.error("Login Username" + object.getUsername() + " is greater than the max-
imum field length [" + MAX_LENGTH + "].");
                                throw new EntitlementValidationPluginException("The login username should not
be greater than [" + MAX_LENGTH + "] characters.");
                        }

                        if(!object.getFirstName().isEmpty() && object.getFirstName().length() > MAX_LENGTH) {
                                LOG.error("Login FirstName" + object.getFirstName() + " is greater than the
maximum field length [" + MAX_LENGTH + "].");
                                throw new EntitlementValidationPluginException("The login first name should
not be greater than [" + MAX_LENGTH + "] characters.");
                        }

                        if(!object.getLastName().isEmpty() && object.getLastName().length() > MAX_LENGTH) {
                                LOG.error("Login LastName" + object.getLastName() + " is greater than the max-
imum field length [" + MAX_LENGTH + "].");
                                throw new EntitlementValidationPluginException("The login last name should not
be greater than [" + MAX_LENGTH + "] characters.");
                        }
                        if(!object.getEmail().isEmpty() && object.getEmail().length() > MAX_LENGTH) {
                                LOG.error("Login Email" + object.getEmail() + " is greater than the maximum
field length [" + MAX_LENGTH + "].");
                                throw new EntitlementValidationPluginException("The login email should not be
greater than [" + MAX_LENGTH + "] characters.");
                        }
                }

                // Retrive the local interface for webservice
                protected AccountManagerWebServiceLocal getAccountManagerWebService() {
                        return (AccountManagerWebServiceLocal) Component.getInstance(AccountManagerWebSer-
vice.class, true);
                }

}
```

Configuring the Plug-in

Perform the following steps to configure a custom Account Validation plug-in:

1. Navigate to **Admin > Config > module > em > login.validation.plugin > classname**.
2. Enter "`com.em.plugin.SampleLoginValidationPlugin`" as the classname.
3. Click **Save**.
4. Navigate to **Entitlement > Accounts > Household**.
5. Click **Add New**.
6. Enter information into the mandatory fields then click **Save**.
7. Click **Add** from the Users field.
8. Enter information into the mandatory fields then click **Save**.



9. Check the log for "`SampleLoginValidationPlugin validate Create invoked`".
10. Open the created account and select the new user.
11. Make all necessary changes.
12. Click **Save**.
13. Check the log for "`SampleLoginValidationPlugin invoked. Validating login update`".

Device Validation Plug-in

Plug-in Architecture

To view the plug-in architecture, see the attached Plug-in Architecture zip file.

Description

A description of this plug-in will be provided ASAP.

To view the methods this interfaces specifies, open the Entitlement Manager folder in “vms_docs-[releasenumber]-[buildnumber]” and select “index.html”.

Example

```
package com.em.plugin;

import org.jboss.seam.Component;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.extend.opencase.em.model.device.Device;
import com.extend.opencase.em.pluggable.validation.EntitlementValidationPlugin;
import com.extend.opencase.em.pluggable.validation.exception.EntitlementValidationPluginException;
import com.extend.opencase.em.webservice.DeviceManagerWebService;
import com.extend.opencase.em.webservice.DeviceManagerWebServiceLocal;
import com.extend.opencase.em.webservice.fault.DeviceManagerWebServiceFault;

/**
* Device Validation Plugin is used to create and validate devices.
*  It is usually called when registering a device.
* Persistence occurs post validation.
*
*/
public class SampleDeviceValidationPlugin implements EntitlementValidationPlugin<Device> {
               private static final transient Logger LOG = LoggerFactory.getLogger(SampleDeviceValidationPlugin.class);
               private static final int MAX_LENGTH = 255;

               /**
               * DeviceValidationPlugin default constructor
               */
               public SampleDeviceValidationPlugin() {
                              super();
               }

               /**
               * Validate that device is not being created with a preset primary key.
                * If device has name validate Name.
               *
                * @param object Device object to validate
               * @throws EntitlementValidationPluginException when validation fails
               */
               @Override
               public void validateCreate(Device object) throws EntitlementValidationPluginException {
                              LOG.info("SampleDeviceValidationPlugin validate create invoked.");

                              if(object.getDevicePk() != null) {
                                             LOG.info("Device has a primary key preset [" + object.getDevicePk() + "].
Please remove primary to create a new device.");
                                             throw new EntitlementValidationPluginException("Device has a primary key pre-
set [" + object.getDevicePk() + "]");
                              }
                              validateName(object);

               }

               /**
               * Validate update of device prior to persisting. Find device by primary key -
                * if not found then device does not exist and validation fails.
                * Validate device has UUID and Primary key since both are
                * generated post persist. If device has a name validate name.
               * @param object Device to validate
               * @throws EntitlementValidationPluginException when validation fails
               */
               @Override
               public void validateUpdate(Device object) throws EntitlementValidationPluginException {
                              if(object.getDevicePk() == null) {
                                             LOG.info("Device does not have a primary key. Cannot update Device.");
                                             throw new EntitlementValidationPluginException("Device does not have a primary
key. Cannot update Device");
```

```
                        }
                        if(object.getUuid() == null) {
                                LOG.info("Device does not have a UUID. Cannot update Device.");
                                throw new EntitlementValidationPluginException("Device does not have a UUID.
Cannot update Device");
                        }
                        //validate this Device already exists
                        DeviceManagerWebServiceLocal deviceWebService = getDeviceManagerWebService();
                        try {
                                Device existingDevice = deviceWebService.findDeviceByPk(object.getDe-
vicePk());
                                if(existingDevice == null) {
                                        LOG.info("Could not find device with primary key [" +
object.getDevicePk() + "], update validation fail.");
                                        throw new EntitlementValidationPluginException("Could not find
device with primary key [" + object.getDevicePk() + "], update validation fail.");
                                }
                        } catch (DeviceManagerWebServiceFault e) {
                                if(LOG.isDebugEnabled()) {
                                        LOG.debug("Could not find device in system", e);
                                }
                                throw new EntitlementValidationPluginException("Could not find device in sys-
tem", e);
                        }

                        validateName(object);

        }

        /**
        * Validate device name
        * @param object device to be validated
        * @throws EntitlementValidationPluginException if validation fails
        */
        private void validateName(Device object) throws EntitlementValidationPluginException {
                        //validate login text field lengths
                        if(object.getName() != null && !object.getName().isEmpty() && object.getName().length() >
MAX_LENGTH) {
                                LOG.error("Device name" + object.getName() + " is greater than the maximum
field length [" + MAX_LENGTH + "].");
                                throw new EntitlementValidationPluginException("The Device name should not be
greater than [" + MAX_LENGTH + "] characters.");
                        }
        }

        /**
        *  Retrieve the local interface for webservice
        * @return AccountManagerWebServiceLocal
        */
        protected DeviceManagerWebServiceLocal getDeviceManagerWebService() {
                        return (DeviceManagerWebServiceLocal) Component.getInstance(DeviceManagerWebService.class,
true);
        }

}
```

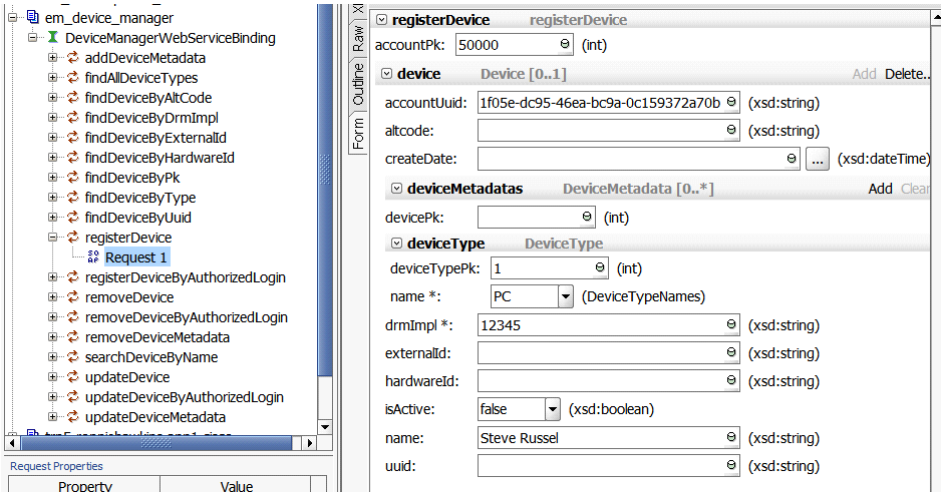
Configuring the Plug-in

Perform the following steps to configure a custom Device Validation plug-in:

1. Navigate to **Admin > Config > module > em > device.validation.plugin > classname.**
2. Enter "`com.em.plugin.SampleDeviceValidationPlugin`" as the classname.
3. Click **Save**.
4. Navigate to **Entitlement > Accounts > Household.**
5. Click **Add New.**

6 Enter information into the mandatory fields then click **Save**.

7 Register the device using "`EntitlementManager.DeviceManagerWebServiceBinding.RegisterDevice`".



8 Check the log for "`SampleDeviceValidationPlugin validate create invoked`".

License Terms Plug-in

Plug-in Architecture

To view the plug-in architecture, see the attached Plug-in Architecture zip file.

Description

A description of this plug-in will be provided ASAP.

Example

```
package com.em.plugin;

import java.util.Calendar;
import java.util.GregorianCalendar;

import javax.xml.datatype.DatatypeConfigurationException;
import javax.xml.datatype.DatatypeFactory;
import javax.xml.datatype.XMLGregorianCalendar;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.extend.opencase.cm.DrmTermsSet;
import com.extend.opencase.cm.PlayReadyDomain;
import com.extend.opencase.cm.PlayReadyDownload;
import com.extend.opencase.cm.WmrmDrmTerms;
import com.extend.opencase.em.credentials.Credentials;
import com.extend.opencase.em.pluggable.PluginInitializationException;
import com.extend.opencase.em.pluggable.license.terms.LicenseTermsPlugin;
import com.extend.opencase.em.pluggable.license.terms.exception.LicenseTermsPluginException;
import com.extend.opencase.webservices.entitlement.AbstractDrmTermsBean;
```

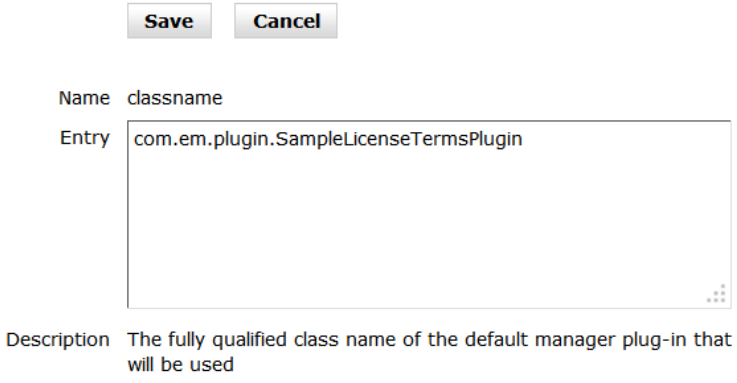
```
/**
* License Terms Plugin called during a License Request to update DrmTermsSet
 *
*/
public class SampleLicenseTermsPlugin  implements LicenseTermsPlugin {
		private static final transient Logger LOG = LoggerFactory.getLogger(SampleLicenseTermsPlugin.class);

		// For issue license start date
		private static final String ISSUEWINDOW_START_TYPE_DATE = "drm.terms.issuewindow.start.type.date";
		// For issue license end date
		private static final String ISSUEWINDOW_END_TYPE_DATE = "drm.terms.issuewindow.end.type.date";
		/**
		* Constructor
		*/
		public SampleLicenseTermsPlugin() {
				super();
		}

		/**
		* Initialize with initData
		 * @param initData
		 * @throws PluginInitializationException
		 */
		@Override
		public void initialize(String initData) throws PluginInitializationException {
				LOG.info("Sample License Terms Plugin initialize invoked.");

		}

		/**
		* Update termsSet. The sample plugin  updates Start and End dates
		 * @param credentials object containing user name/password and affiliate for authentication.
		* @param termsSet termSet from DrmProfile to update
		*
		 * @throws LicenseTermsPluginException
		 */
		@Override
		public void updateDrmTermsSet(Credentials credentials, DrmTermsSet termsSet) throws LicenseTermsPluginExcep-
tion {
				AbstractDrmTermsBean drmTermsBean = termsSet.getDrmTermsBean();
				if(drmTermsBean instanceof PlayReadyDownload) {
						PlayReadyDownload drmTerms = (PlayReadyDownload) drmTermsBean;
						drmTerms.setIssueWindowStart(ISSUEWINDOW_START_TYPE_DATE);
						drmTerms.setIssueWindowStartDate(getXMLGregorianCalendar(10));
						drmTerms.setIssueWindowEnd(ISSUEWINDOW_END_TYPE_DATE);
						drmTerms.setIssueWindowEndDate(getXMLGregorianCalendar(30));
				} else if(drmTermsBean instanceof PlayReadyDomain) {
						PlayReadyDomain drmTerms = (PlayReadyDomain) drmTermsBean;
						drmTerms.setIssueWindowStart(ISSUEWINDOW_START_TYPE_DATE);
						drmTerms.setIssueWindowStartDate(getXMLGregorianCalendar(10));
						drmTerms.setIssueWindowEnd(ISSUEWINDOW_END_TYPE_DATE);
						drmTerms.setIssueWindowEndDate(getXMLGregorianCalendar(30));
				} else if(drmTermsBean instanceof WmrmDrmTerms) {
						WmrmDrmTerms drmTerms = (WmrmDrmTerms) drmTermsBean;
						drmTerms.setIssueWindowStart(ISSUEWINDOW_START_TYPE_DATE);
						drmTerms.setIssueWindowStartDate(getXMLGregorianCalendar(10));
						drmTerms.setIssueWindowEnd(ISSUEWINDOW_END_TYPE_DATE);
						drmTerms.setIssueWindowEndDate(getXMLGregorianCalendar(30));
				}
		}

		/**
		* Adds the specified (signed) amount of time to the current time of calendar
		*
		 * @param amount in minutes
		* @return time of calendar that is before or after the specified amount of time
		* @throws LicenseTermsPluginException
		*/
		private XMLGregorianCalendar getXMLGregorianCalendar(int amount) throws LicenseTermsPluginException {
				GregorianCalendar calendar = new GregorianCalendar();
				calendar.add(Calendar.MINUTE, amount);
				try {
```

```
                                        return DatatypeFactory.newInstance().newXMLGregorianCalendar(calendar);
                                } catch(DatatypeConfigurationException e) {
                                        throw new LicenseTermsPluginException("Error when creating Xml GregorianCalen-
dar time", e);
                                }
                }
}
```

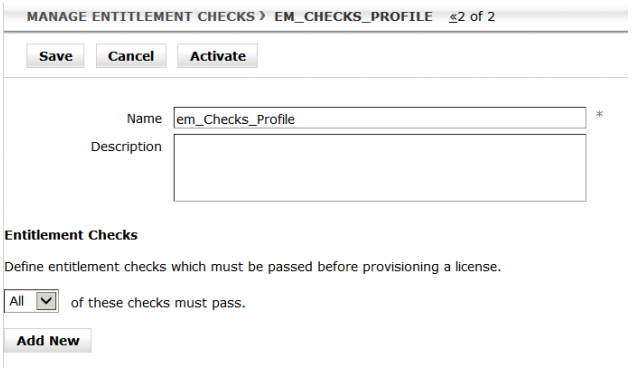
Configuring the Plug-in

Perform the following steps to configure a custom License Terms plug-in:

1. Navigate to **Admin > Setup > Configuration > modules > em > license.terms.plugin > classname.**
2. Enter "`com.em.plugin.SampleLicenseTermsPlugin`" as the classname.



3. Click **Save**.
4. Create an entitlement check profile:
   a. Navigate to **Entitlement > Productization > Profiles > Entitlement Checks**.
   b. Click **Add New.**
   c. Enter a name and description.
   d. Click **Create & Edit**.
   e. Click **Save** then **Activate.**



5 Create a DRM profile:

a Navigate to **Entitlement > Productization > Profiles > DRM Profiles.**

b Click **Add New.**

c Enter a name and description.

d Click **Create & Edit**.

e Click the PlayReady License check box.

f Click the PlayReady License link.

g Select the minimum security (any available value).

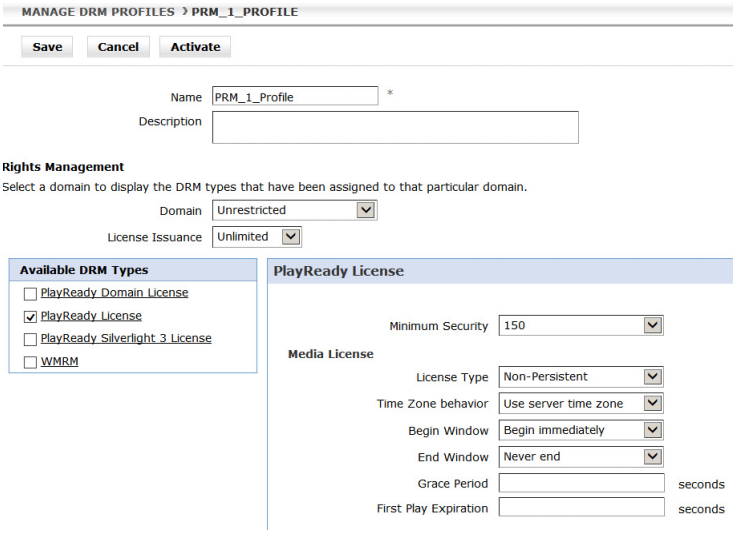
h Select “Non-Persistent” as the license type.

i Select “User server time zone” as the time zone.

j Select “Begin Immediately” as the begin window.

k Select “Never end” as the end window.

l Click **Save** then **Activate**.



6 Create a policy:

a Navigate to **Entitlement > Productization > Policies.**

b Click **Add New.**

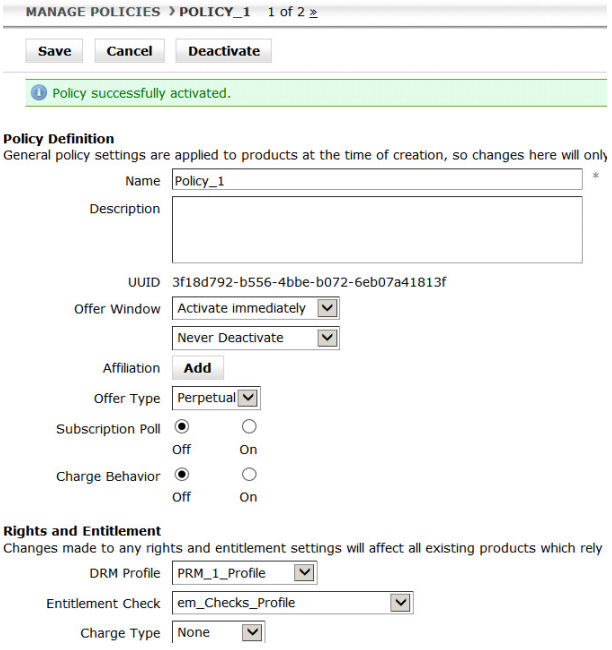
c Enter a name and description.

d Click **Create & Edit**.

e Select the entitlement check profile created in Step 4 from the Entitlement Check drop-down menu.

f Select the DRM profile created in Step 5 from the DRM Profile drop-down menu.

g Click **Save** then **Activate.**



7 Create a new bundle.

a Navigate to **Metadata > Bundles**.

b Click **Add New** and select “Logical Video”.

c Enter a name and description.

d Click **Create & Edit.**

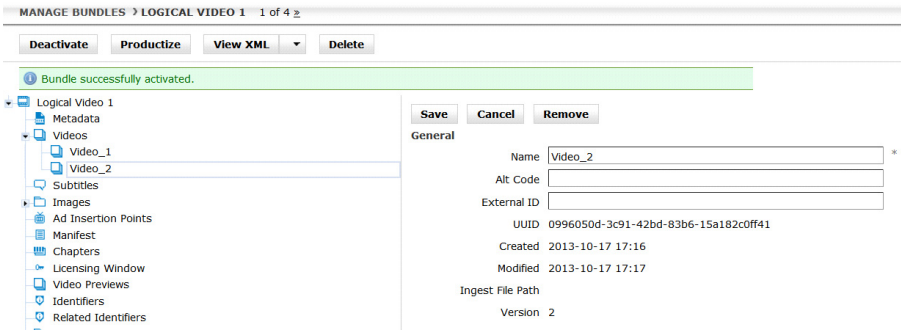
e Navigate to **Metadata > Videos.**

f Click **Add New** and select “Physical Asset Video or Manifest”

g Enter a name and description.

h Click **Create & Edit.**

i Scroll down to the content ID field and enter a valid content ID.

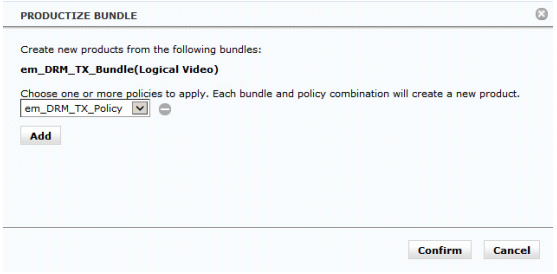


j Click **Save** then **Activate.**

k Click **Productize**.

l Click Add and select the policy created above with the entitlement check profile.

m Click **Confirm**.



8 Create an account.

a Navigate to **Entitlement > Accounts > Household**.

b Click **Add New**.

c Enter information into all mandatory fields.

d Click **Save.**

9 Create an entitlement using the following web method:

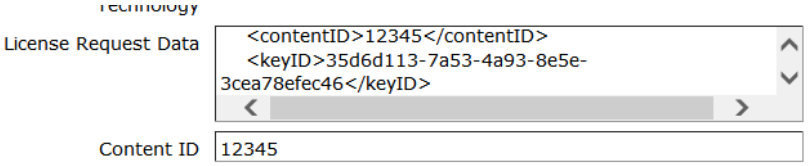
`RightsLockerWebService.createEntitlementByAccountIdAndProductId(<AccountUUIDOfTheAccountCreatedAbove>,<ProductUUIDOfTheProductCreatedAbove>)`

10 Create encryption data in the physical asset video or manifest.

11 Execute `PlayReadyProviderService.GetEncryptInfo(inputXml)`.

12 Take note of the keyID in the License Request Data of the physical asset.

13 Make a note of the keyId in the License Request Data of the physical asset.



14 Retrieve the license request fro the product created above using the `keyId`, `PhysicalAssetUUID`, and the `EntitlementUUID`.

15 Check the log to ensure that the DRM transaction(create/update/delete/commit) is logged.

16 Check the log for "`Sample License Terms Plugin Initialize invoked`".

DRM Key Provider Plug-in

Plug-in Architecture

To view the plug-in architecture, see the attached Plug-in Architecture zip file.

Description

A description of this plug-in will be provided ASAP.

To view the methods this interfaces specifies, open the Entitlement Manager folder in "vms_docs-[releasenumber]-[buildnumber]" and select "index.html".

Example

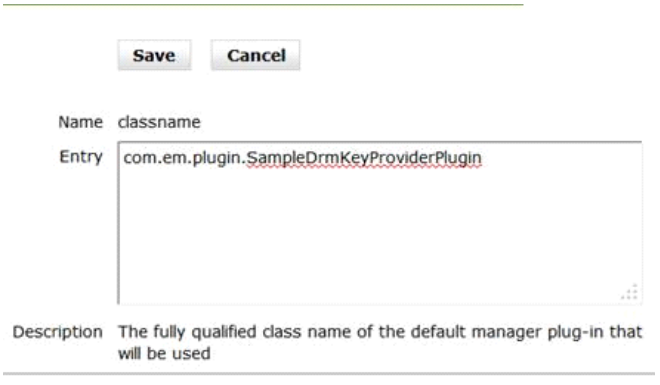
```
package com.em.plugin;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.extend.opencase.cm.Product;
import com.extend.opencase.em.credentials.Credentials;
import com.extend.opencase.em.model.account.Login;
import com.extend.opencase.em.pluggable.PluginInitializationException;
import com.extend.opencase.em.pluggable.drm.key.exception.DrmKeyProviderPluginException;
import com.extend.opencase.em.pluggable.drm.key.DrmKeyProviderPlugin;
import com.extend.opencase.esb.services.drm.packager.interfaces.PackagingHeader;
import com.extend.opencase.esb.services.drm.packager.interfaces.WmrmPackagingHeader;

/**
* SampleDrmKeyProviderPlugin is mainly used to return a DRM content key if
*  valid parameters are passed in during an entitlement check.
*
 */
public class SampleDrmKeyProviderPlugin implements DrmKeyProviderPlugin {
               private static final transient Logger LOG = LoggerFactory.getLogger(SampleDrmKeyProviderPlugin.class);

               private static final String SAMPLE_DRM_KEY="Sample Key";

               /**
               * Initialize DrmKeyProvider Plugin
               * @param initData
                */
               @Override
```

```
        public void initialize(String initData) throws PluginInitializationException {
                if (null == initData || initData.isEmpty()) {
                        LOG.info("Sample Drm Key Provider Plugin Initialize invoked with no init-
Data");
                        return;
                } else if(!initData.isEmpty()){
                        LOG.info("Sample Drm Key Provider Plugin Initialize invoked [" + initData +
"].");
                }
        }

        /**
        * Sample Plugin returns sample key only if packaging header
         * is an instance of PlayReady, otherwise return null;
        * @param product Product that is being checked
        * @param login Login
         * @param Credentials
        * @param packagingHeader
        *
         */
        @Override
        public String requestDrmKey(Product product, Login login, Credentials credentials, PackagingHeader packag-
ingHeader)
                        throws DrmKeyProviderPluginException {
                LOG.info("Sample Drm Key Provider Plugin requestDrmKey invoked");
                if(packagingHeader instanceof WmrmPackagingHeader) {
                        return SAMPLE_DRM_KEY;
                }
                return null;
        }

}
```

Configuring the Plug-in

Perform the following steps to configure a custom DRM Key Provider plug-in:

1. Navigate to **Admin > Setup > Configuration > modules > em > drm.key.provider.plugin > classname.**
2. Enter "`com.em.plugin.SampleDrmKeyProviderPlugin`" as the classname.

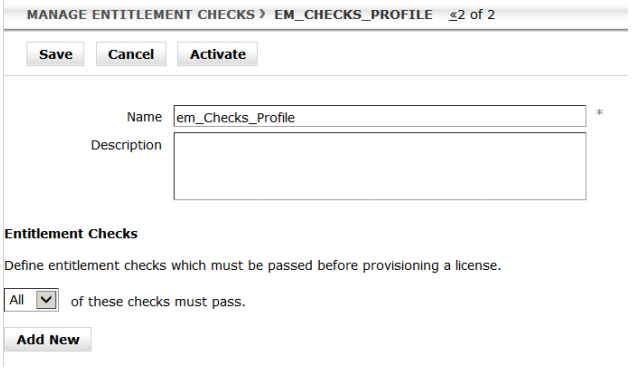


3. Click **Save.**
4. Create an entitlement check profile:
   a. Navigate to **Entitlement > Productization > Profiles > Entitlement Checks**.
   b. Click **Add New**.

c Enter a name and description.

d Click **Create & Edit**.

e Click **Save** then **Activate.**



5 Create a DRM profile:

a Navigate to **Entitlement > Productization > Profiles > DRM Profiles.**

b Click **Add New**.

c Enter a name and description.

d Click **Create & Edit**.

e Click the PlayReady License check box.

f Click the PlayReady License link.

g Select the minimum security (any available value).

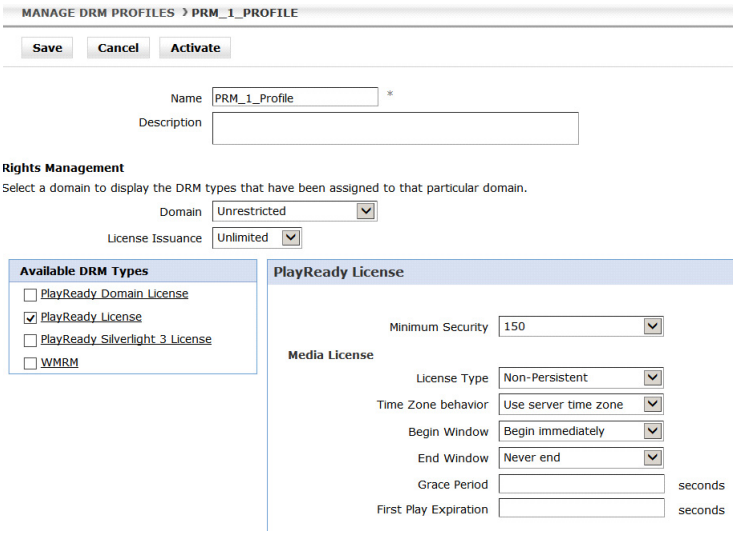
h Select “Non-Persistent” as the license type.

i Select “User server time zone” as the time zone.

j Select “Begin Immediately” as the begin window.

k Select “Never end” as the end window.

l Click **Save** then **Activate**.



6 Create a policy:

a Navigate to **Entitlement > Productization > Policies**.

b Click **Add New.**

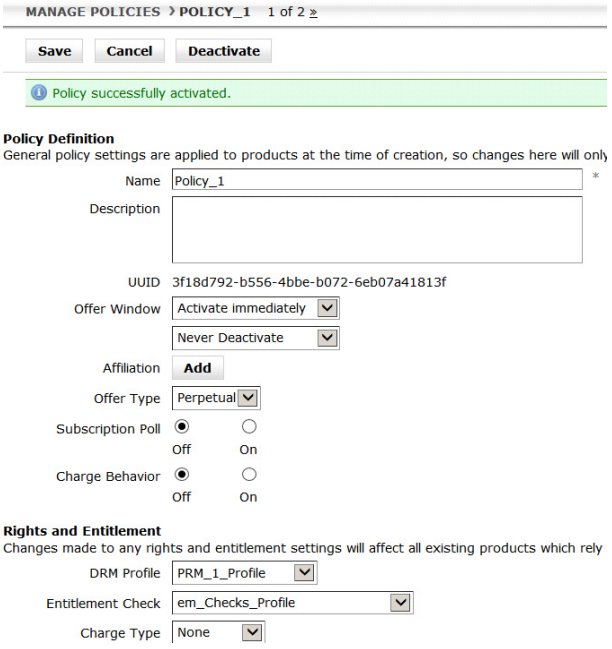
c Enter a name and description.

d Click **Create & Edit**.

e Select the entitlement check profile created in Step 4 from the Entitlement Check drop-down menu.

f Select the DRM profile created in Step 5 from the DRM Profile drop-down menu.

g Click **Save** then **Activate.**



7 Create a new bundle.

a Navigate to **Metadata > Bundles**.

b Click **Add New** and select “Logical Video”.

c Enter a name and description.

d Click **Create & Edit**.

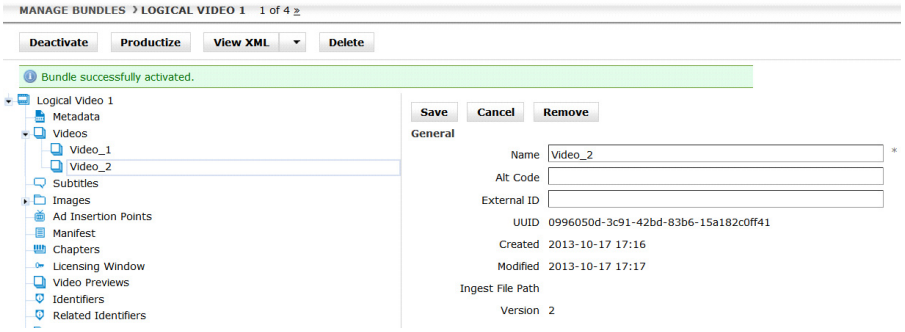
e Navigate to **Metadata > Videos**.

f Click **Add New** and select “Physical Asset Video or Manifest”

g Enter a name and description.

h Click **Create & Edit.**

i Scroll down to the content ID field and enter a valid content ID.

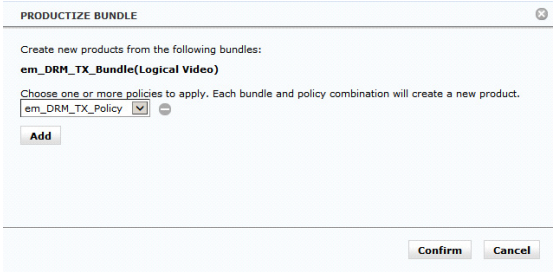


j Click **Save** then **Activate**.

k Click **Productize**.

l Click **Add** and select the policy created above with the entitlement check profile.

m Click **Confirm**.



8 Create an account.

a Navigate to **Entitlement > Accounts > Household**.

b Click **Add New.**

c Enter information into all mandatory fields.

d Click **Save**.

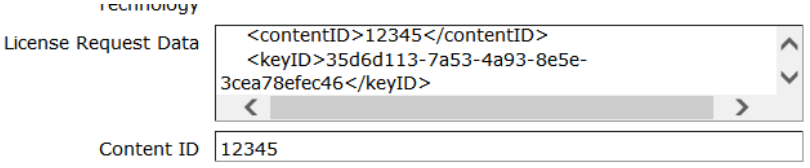
9 Create an entitlement using the following web method:
`RightsLockerWebService.createEntitlementByAccountIdAndProductId(<AccountUUIDOfTheAccountCreatedAbove>,<ProductUUIDOfTheProductCreatedAbove>)`

10 Create encryption data in the physical asset video or manifest.

11 Execute `PlayReadyProviderService.GetEncryptInfo(inputXml)`.

12 Take note of the keyID in the License Request Data of the physical asset.

13 Make a note of the keyId in the License Request Data of the physical asset.



14 Retrieve the license request fro the product created above using the `keyId`, `PhysicalAssetUUID`, and the `EntitlementUUID`.

15 Check the log to ensure that the DRM transaction(create/update/delete/commit) is logged.

16 Check the log for "`Sample Drm Key Provider Plugin Initialize invoked`".

17 Check the log for "`Sample Drm Key Provider Plugin requestDrmKey invoked`".

Chapter 7

Creating Custom ESB Plug-ins

This chapter includes information about creating the following custom ESB plug-in for each:

ESB

Custom ESB Service

Plug-in Architecture

To view the plug-in architecture, see the attached Plug-in Architecture zip file.

Description

For information on ESBs, please see the Videoscape Media Suite 4.1 Developer Guide.

Building the Plug-in

Perform the following steps to build a custom ESB plug-in:

1. Build an unsigned JAR file.
   a. Open the command prompt.
   b. Change the directory to `<my-project-root>/<custom-esb-plugin>`
   c. Execute `mvn -PunsignJar clean install`
2. Build an ESB file.
   ```
   <my-project-root>/<custom-esb-plugin>
   mvn jboss-packaging:esb
   ```
3. Navigate to **Configure > Custom File Deployment > Add New**.
4. Deploy the JAR file into the Workflow Service.
   a. Enter a name in the **Name** field.
   b. Set the target path to `deploy/WorkflowService.ear/lib`
   c. Select Workflow Service from the **Target Module** drop-down menu.
   d. Check **Place Inside Module Directory**.
   e. Upload the custom JAR file `<custom-esb-plugin-1.25.jar>`
   f. Click **Save**.

g Re-open the file then click **Activate**.

5 Deploy the JAR file into the Content Processor.

a Click Add New.

b Enter a name in the **Name** field.

c Set the target path to `deploy/ContentProcessor.ear/lib`

d Select Content Processor from the **Target Module** drop-down menu.

e Check "Place inside module directory".

f Upload the custom JAR file `<custom-esb-plugin-1.25.jar>`

g Click **Save**.

h Re-open the file then click **Activate**.

6 Deploy the ESB into JBoss/ESB Messaging.

a Click **Add New**.

b Enter a name.

c Set the target path to `deploy/`

d Select JBoss/ESB Messaging from the **Target Module** drop-down menu.

e Upload the ESB file `<custom-esb-plugin-1.25.esb>`

f Click **Save**.

g Re-open the file then click **Activate**.

7 Restart VMS.

Configuring the Plug-in

8 Create an Action Template

a Navigate to **Workflow > Setup > Action Templates > Add New**.

b Enter a name.

c Select `OC_ESB_Processor_Custom_Helloworld:CustomHelloworldService` from the **ESB Service** drop-down menu.

d Click **Save** then **Activate**.

Triggering the Plug-in

9 Create an Import Workflow template.

a Enter a name.

b Browse to the custom_Helloworld_WF.par file.

c Click **Save** then **Activate**.

10 Create an Action Profile.

a Navigate to **Workflow > Manage > Action Profiles > Add New.**

b Enter a name.

c Click **Create & Edit**.

d Click **Save** then **Activate**.

11 Create a Workflow Definition.

a Navigate to **Workflow > Manage > Workflow Definitions > Add New**.

b Enter a name.

c Click **Create & Edit**.

d Open the **Configure** tab.

e Click **<<ESB Service>>.**

f Select the new Action Profile in the **Processor** drop-down menu.

12 Click **Save**.

13 Click **Deploy**.

14 Click **Activate**.

15 Drop an image into the Hot Folder.

Expected Results

Check that the Physical Asset is listed by navigating to **Metadata > Components**.

Chapter 8

Creating Custom Producer Plug-ins

This chapter includes information about creating the following custom Producer plug-ins:

Producer

Prerequisites

Follow the instructions below to create Producer tabs:

1. Install the parent POM file into the local Maven repository.
2. Generate the plug-in archetype.
3. Create a "Project Root" folder on a local computer to house the custom plug-in(s).
4. Build the plug-in files and place in the "Project Root" folder.

```
cd <Project root>
mvn archetype:generate
-DarchetypeGroupId=com.xxx.vms.plugins
-DarchetypeArtifactId=vms-producer-plugins-archetype
-DarchetypeVersion=x.x-xxxxxx (plugins version)
-DgroupId=com.xxx.vms
-DartifactId=custom-producer-plugin
-Dversion=1.25
```

5. Build a signed JAR file.
6. Navigate to **Configuration > Custom File Deployment > Add New.**

Follow the steps below when using a 5.1.x version of VMS

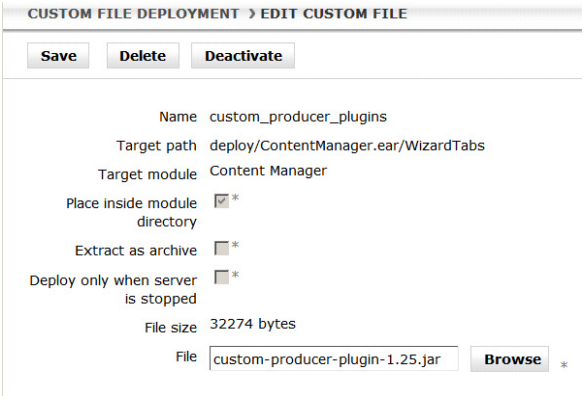
1. Enter a name.
2. Set the target path to: deploy/ContentManager.ear/WizardTabs.

3 Set the target module to: Content Manager.

4 Check the box “Place inside module directory”.

5 Click **Browse** and select the “`custom_producer_plugins.jar`” file

6 Click **Save** then **Activate**.



Follow the steps below when using a 5.5 version of VMS

1 Deploying custom Producer wizard tab plug-ins.

   a Enter a name.

   b Set the target path to: deploy/ContentManager.ear/WizardTabs.

   c Set the target module to: Producer.

   d Uncheck the box “Place inside module directory”.

   e Browse and select the “`custom_producer_wizard_tabs_plugin-1.25.jar`” file.

   f Click **Save** then **Activate**.

2 Restart VMS.

3 Deploying custom Metadata Search plug-ins.

   a Enter a name.

   b Set the target path to: deploy/ContentManager.ear/lib.

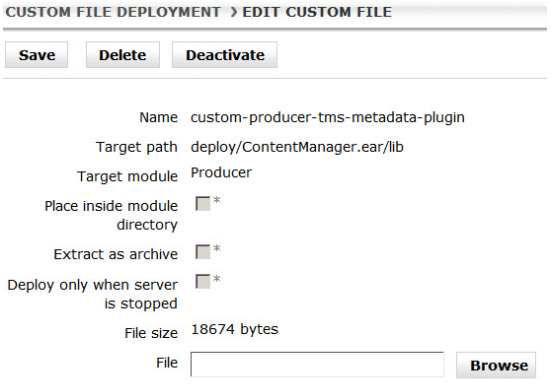
   c Set the target module to: Producer.

d Uncheck the box “Place inside module directory”.

e Browse and select the “`custom_producer_tms_metadata_plugin-1.25.jar`” file.

f Click **Save** then **Activate**.

4 Restart VMS.



Custom Wizard Tabs

The custom wizard tab plug-in adds tabs within a wizard for specific bundle types. Common tabs for Metadata, Images, Videos, Categories, and Availabilities are automatically configured in Producer. Support for any other custom tab can be implemented and deployed using the information contained in this chapter. Tabs are constructed using two components: an XHTML page and a correponding java action bean. Included are examples of custom tabs that can be used as a refrence when creating additional tabs. The following tabs are included in this chapter:

- Identifier tab
- Logical Video tab
- Metadata tab
- Subtitle tab

Metadata Plug-in Interface Methods to Implement

All tabs are required to extend the `WizardTab` interface. This interface includes the following methods which dictate actions to be performed when a user navigates in and out of the tab.

```
    /**
         * Called each time when tab is selected on UI to initialize the tab
     */
        void onTabSelected();

    /**
         * Called each time when leave the tab
    */
void onTabLeave();
```

If any of the built tabs must be customized further, the corresponding interfaces for those tabs that extend the base `WizardTab` interface can be used.

METADATA WIZARD INTERFACE

```
        /**
         * Returns list of available locales for the dropdown list to create a new metadata
         *
         * @return list of locales
         */
        List<Locale> getAvailableLocales();

        /**
         * Returns of all metadata for current bundle
         *
         * @return list of metadata
         */
        List<AbstractComponent> getMetadataList();

        /**
         * Create a new metadata by selected locale
         *
         * @param locale locale for which new metadata will be created
         */
        void addMetadata(Locale locale);

        /**
         * Updates selected metadata object
         */
        void save();

        /**
         * Removes selected metadata object
         */
        void delete();

        /**
         * Revert changes on selected metadata object
         */
        void revert();

        /**
         * Returns form config object that define how many fields of component should be shown on
UI
         *
         * @return form config object
         */
        FormConfig getFormConfig();

        /**
         * Returns compound form elements (common entities, custom attributes)
         *
         * @return compound form elements (common entities, custom attributes)
         */
        CompoundFormElements getCompoundFormElements();
```

```
        /**
         * Gets selected component as metadata
         *
         * @return selected metadata
         */
Metadata getMetadata();
```

IMAGE WIZARD INTERFACE

```
        /**
         * Updates selected image object
         */
        void save();

        /**
         * Removes selected image object
         */
        void delete();

        /**
         * Revert changes on selected image object
         */
        void revert();

        /**
         * Returns current image folder
         *
         * @return current image folder or [null] if no image selected
         */
        BundleItem getComponentFolder();

        /**
         * Returns current folder
         *
         * @return current image folder or [null] if no folder selected
         */
        BundleItem getFolder();

        /**
         * Returns form config object that define how many fields of component should be shown on
UI
         *
         * @return form config object
         */
        FormConfig getFormConfig();

        /**
         * Returns compound form elements (common entities, custom attributes)
         *
         * @return compound form elements (common entities, custom attributes)
         */
        CompoundFormElements getCompoundFormElements();
```

```
        /**
         * Adds uploaded image to bundle
         *
         * @param fileUploadBean Contains the details of the uploaded image
         */
void addImageToBundle(FileUploadBean fileUploadBean);
```

VIDEO WIZARD INTERFACE

```
/**
         * Associate selected video object to the bundle item
         */
        void addSelected();

        /**
         * Updates selected video object
         */
        void save();

        /**
         * Removes or disassociates selected video object
         */
        void delete();

        /**
         * Revert changes on selected video object
         */
        void revert();
```

CATEGORY WIZARD INTERFACE

```
        /**
         * Saves the updated metadata
         */
        void save();

        /**
         * Revert changes on categories of metadata
         */
void revert();
```

AVAILABILITY WIZARD INTERFACE

```
        /**
         * Updates selected availability object
         */
        void save();

        /**
         * Removes or Disassociates selected availability object
         */
```

```
        void delete();

        /**
         * Revert changes on selected availability object
         */
        void revert();

        /**
         * Returns form config object that define how many fields of component should be shown on
UI
         *
         * @return form config object
         */
FormConfig getFormConfig();
```

Abstract Wizard Tab Classes

Abstract classes include methods to help build custom tabs.These abstract classes are separated into action classes that corresponding to different aspects of the tab, including common actions, search actions, and tree actions.

ABSTRACTWIZARDTAB.JAVA

This is the base abstract class that contains methods which will retrieve instances of all components needed to build a tab.

```
/**
         * Gets Selected component Name
         *
         * @return component Name
         */
        public abstract String getSelectedComponentName();

        /**
         * Gets selected bundle
         *
         * @return Currently selected Bundle.
         */
        protected Bundle getCurrentBundle()

        /**
         * Gets Component Webservice
         *
         * @return ComponentWebService instance.
         */
        protected ComponentWebServiceLocal getComponentWebService()

        /**
         * Gets Search Web Service
         *
         * @return SearchWebService instance.
         */
        protected SearchWebServiceLocal getSearchWebService() {
```

```
/**
 * Gets Common Entity Web Service
 *
 * @return CommonEntityWebService instance.
 */
protected CommonEntityWebServiceLocal getCommonEntityWebService()

/**
 * Gets Faces Messages
 *
 * @return FacesMessages instance.
 */
protected FacesMessages getFacesMessages() {


/**
 * Gets Selected Wizard Template Tab
 *
 * @return selected WizardTemplateTab instance.
 */
protected WizardTemplateTab getSelectedWizardTemplateTab()

/**
 * Gets Character Converter Service
 *
 * @return CharacterConverterService instance.
 */
protected CharacterConverterService getCharacterConverterService()

/**
 * Gets CDN Mapping Service
 *
 * @return CdnMappingService instance.
 */
protected CdnMappingService getCdnMappingService()

/**
 * Gets Category Service
 *
 * @return CategoryService instance.
 */
protected CategoryService getCategoryService()

/**
 * Gets Dummy bundle for Bulk Edit
 *
 * @return Currently selected Bundle.
 */
protected Bundle getDummyBundle()

/**
 * Gets the list of bundles selected for Bulk Edit
```

```
         *
         * @return Currently selected Bundle.
         */
        @SuppressWarnings("unchecked")
protected List<Bundle> getSelectedBundles()
```

ABSTRACTWIZARDTABACTIONS.JAVA

This class contains methods to initialize all bundle information that is based on the selected bundle type and corresponding component type specified within the bundle configuration XML file. This class will also generate the component form elements using the dynamic form; it will then filter based on the fields specified in the bundle configured XML file.

Note This class extends AbstractWizardTab.java

ABSTRACTWIZARDCOMMONACTIONS.JAVA

This class contains methods to save, deleted, and revert updates.

Note This class extends AbstractWizardTabActions.java

ABSTRACTWIZARDTABTREEACTIONS.JAVA

This class contains methods that correspond to actions that are performed within tree nodes (if one exists in the custom tab). Methods include initializing the tree, processing the tree node selection, refreshing the tree, and updating values in the tree.

Note This class extends AbstractWizardCommonActions.java

ABSTRACTWIZARDSEARCHACTIONS.JAVA

This class contains methods for search actions based on the component(s) specified in the bundle template configuration. Also contained are methods for searching and associating search items to the specified bundle.

Note This class extends AbstractWizardTabTreeAction.java

Bundle Wizard Configuration Settings

After deploying a custom wizard plug-in, the bundle wizard XML (for the corresponding bundle wizard configuration) must be configured to retrieve those tabs in the wizard.

```
<tab order="1" label="bundle.folder.type.metadata">
                <implementation>actionbean</implementation>
                <presentation>xhtmlFile</presentation>
                <components>
                        <YOUR_COMPONENT_IN_THE_TAB>
               <field name="name" />
```

```
                                <field name="altCode" />
        </ YOUR_COMPONENT_IN_THE_TAB >
                </components>
         </tab>
```

The “implementation” tag is used to store the name of the action bean, while the “presentation” tag is used to store the name of the XHTML path.

Note The presentation tag should contain the full path to the XHTML file.

Logical Video Wizard Tab

Plug-in Architecture

To view the plug-in architecture, see the attached Plug-in Architecture zip file.

Description

The logical video wizard is a feature that allows for customization of bundles. Customers can manage their own objects and XHTML files, which will be integrated with Producer at the time of compilation and deployment. All required components must be added to a bundle for the bundle wizard to work successfully.

To view the methods this interfaces specifies, open the Content Manager folder in “vms_docs-[releasenumber]-[buildnumber]” and select “index.html”.

Example

The following is a sample Logical Video Wizard Tab:

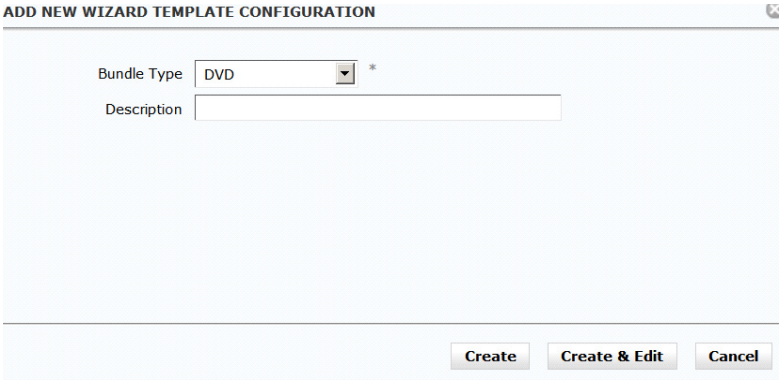
```
package com.cisco.tab.identifier;

import java.util.ArrayList;
import java.util.List;

import org.jboss.seam.ScopeType;
import org.jboss.seam.annotations.Name;
import org.jboss.seam.annotations.Scope;

import com.extend.opencase.cm.action.component.template.CompoundFormElements;
import com.extend.opencase.cm.model.component.AbstractComponent;
import com.extend.opencase.cm.model.component.ComponentType;
import com.extend.opencase.cm.model.component.bundle.BundleItem;
import com.extend.opencase.cm.model.component.identifier.AbstractIdentifier;
import com.extend.opencase.cm.webservice.fault.ComponentParameterFault;
import com.extend.opencase.cm.webservice.fault.ComponentWebserviceFault;
import com.extend.opencase.pt.action.wizard.tab.common.AbstractWizardTabSearchActions;
import com.extend.opencase.pt.action.wizard.tab.common.ComponentWebServiceSearchResult;

@Name("identifierWizardTabAction")
@Scope(ScopeType.PAGE)
public class IdentifierWizardTabAction extends AbstractWizardTabSearchActions {

          private static final String IDENTIFIERS_FOLDER = "Identifiers";

          private static final long serialVersionUID = 1L;

          private List<ComponentType> availableComponentTypes;
```

```
	public IdentifierWizardTabAction() {
		super();
	}

	@Override
	protected void init() {
		super.init();

		initComponentsMap(AbstractIdentifier.class);
		initTree();

		selectFirstFolder();
	}

	@Override
	protected void initComponentsMap(Class<?> componentClass) {
		super.initComponentsMap(componentClass);

		initAvailableComponentTypes();
	}

	@Override
	public void save() {
		super.save();
		refreshTreeAfterUpdate();
	}

	@Override
	public void delete() {
		super.delete();
		refreshTreeAfterDelete(AbstractIdentifier.class, "mainAreaDiv:identifierTab");
	}

	@Override
	public CompoundFormElements getCompoundFormElements() {
		refreshTabPanel("mainAreaDiv:identifierTab");
		return super.getCompoundFormElements();
	}

	public void addSelected() {
		addSelected(AbstractIdentifier.class, "mainAreaDiv:identifierTab");
	}

	public void add(ComponentType componentType) {
		AbstractIdentifier newIdentifier = (AbstractIdentifier) componentType.getInstanceClass();
		newIdentifier.setName(componentType.getName());

		try {
			AbstractIdentifier createdIdentifier = (AbstractIdentifier)
getComponentWebService().createComponent(newIdentifier);

			getComponentWebService().addComponentToBundle(createdIdentifier.getComponentPk(),
getCurrentBundle().getComponentPk(),
					folder.getBundleItemPk());

			initComponentsMap(AbstractIdentifier.class);
			initTree();
			selectFolder(folder);
		} catch(ComponentParameterFault componentParameterFault) {
			addFailedMessage(componentParameterFault, "Failed to create new identifier component.");
		} catch(ComponentWebserviceFault componentWebserviceFault) {
			addFailedMessage(componentWebserviceFault, "Failed to create new identifier component.");
		}

}
```

Building the Plug-in

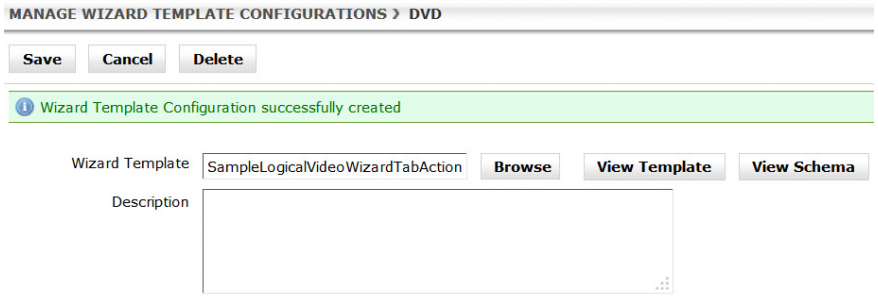
Follow the steps below to build a custom Logical Video Wizard Tab in 5.1.x versions of VMS:

1. Navigate to **Producer > Setup > Wizard Template Configuration.**
2. Click **Add New**.
3. Select DVD as the bundle type on the "Add New Wizard Template Configuration" window.
4. Click **Create & Edit**.



Follow the steps below to build a custom Logical Video Wizard Tab in a 5.5 version of VMS:

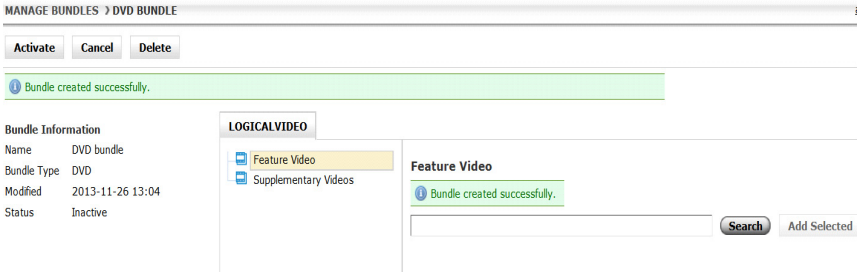
1. Navigate to **Producer > Setup > Wizard Template Configuration**.
2. Select DVD as the bundle type and BundleWizard as the template type on the "Add New Wizard Template Configuration" window.
3. Click **Create & Edit.**
4. To add a Wizard Template, browse to the "`SampleLogicalVideoWizardTabAction.xml`" file.
5. Click **Save**.



6. Navigate to **Producer > Manage Bundles > Add New > DVD**.
   a. Enter a name.

b Click **Create & Edit.**

Note The new DVD bundle must contain the bundle information and “LOGICAL VIDEO” tab.



Metadata Wizard Tab

Plug-in Architecture

To view the plug-in architecture, see the attached Plug-in Architecture zip file.

Description

The Metadata tab allows users to add, remove or update metadata for a logical video bundle.By default, metadata for a new bundle does not exist and must be added by selecting a locale.

To view the methods this interfaces specifies, open the Content Manager folder in “vms_docs-[releasenumber]-[buildnumber]” and select “index.html”.

Example

The following is a sample Metadata Wizard Tab:

```
package com.cisco.tab.identifier;

import java.util.ArrayList;
import java.util.List;

import org.jboss.seam.ScopeType;
import org.jboss.seam.annotations.Name;
import org.jboss.seam.annotations.Scope;

import com.extend.opencase.cm.action.component.template.CompoundFormElements;
import com.extend.opencase.cm.model.component.AbstractComponent;
import com.extend.opencase.cm.model.component.ComponentType;
import com.extend.opencase.cm.model.component.bundle.BundleItem;
import com.extend.opencase.cm.model.component.identifier.AbstractIdentifier;
import com.extend.opencase.cm.webservice.fault.ComponentParameterFault;
import com.extend.opencase.cm.webservice.fault.ComponentWebserviceFault;
import com.extend.opencase.pt.action.wizard.tab.common.AbstractWizardTabSearchActions;
import com.extend.opencase.pt.action.wizard.tab.common.ComponentWebServiceSearchResult;

@Name("identifierWizardTabAction")
@Scope(ScopeType.PAGE)
public class IdentifierWizardTabAction extends AbstractWizardTabSearchActions {
```

```
	private static final String IDENTIFIERS_FOLDER = "Identifiers";

	private static final long serialVersionUID = 1L;

	private List<ComponentType> availableComponentTypes;

	public IdentifierWizardTabAction() {
		super();
	}

	@Override
	protected void init() {
		super.init();

		initComponentsMap(AbstractIdentifier.class);
		initTree();

		selectFirstFolder();
	}

	@Override
	protected void initComponentsMap(Class<?> componentClass) {
		super.initComponentsMap(componentClass);

		initAvailableComponentTypes();
	}

	@Override
	public void save() {
		super.save();
		refreshTreeAfterUpdate();
	}

	@Override
	public void delete() {
		super.delete();
		refreshTreeAfterDelete(AbstractIdentifier.class, "mainAreaDiv:identifierTab");
	}

	@Override
	public CompoundFormElements getCompoundFormElements() {
		refreshTabPanel("mainAreaDiv:identifierTab");
		return super.getCompoundFormElements();
	}

	public void addSelected() {
		addSelected(AbstractIdentifier.class, "mainAreaDiv:identifierTab");
	}

	public void add(ComponentType componentType) {
		AbstractIdentifier newIdentifier = (AbstractIdentifier) componentType.getInstanceClass();
		newIdentifier.setName(componentType.getName());

		try {
			AbstractIdentifier createdIdentifier = (AbstractIdentifier)
getComponentWebService().createComponent(newIdentifier);

			getComponentWebService().addComponentToBundle(createdIdentifier.getComponentPk(),
getCurrentBundle().getComponentPk(),
					folder.getBundleItemPk());

			initComponentsMap(AbstractIdentifier.class);
			initTree();
			selectFolder(folder);
		} catch(ComponentParameterFault componentParameterFault) {
			addFailedMessage(componentParameterFault, "Failed to create new identifier component.");
		} catch(ComponentWebserviceFault componentWebserviceFault) {
			addFailedMessage(componentWebserviceFault, "Failed to create new identifier component.");
		}

}
```

Building the Plug-in

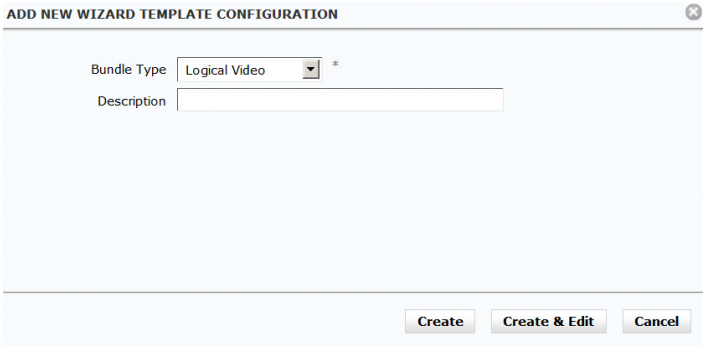
Follow the steps below to create a custom Metadata Wizard tab in 5.1.x version of VMS:

1 Navigate to **Producer > Setup > Wizard Template Configuration**.

2 Click **Add New.**

3 Select Logical Video as the bundle type in the "Add New Wizard Template Configuration" window.

4 Click **Create & Edit.**



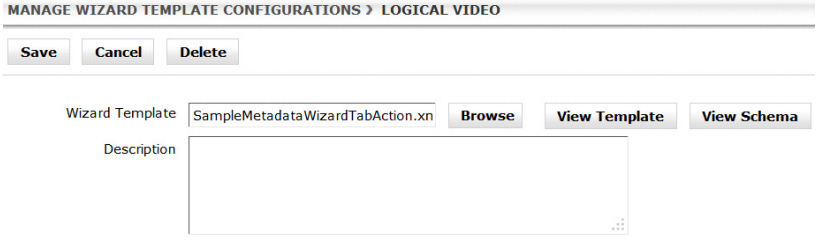
Follow the steps below to create a custom Metadata Wizard tab in a 5.5 version of VMS:

5 Navigate to **Producer > Setup > Wizard Template Configuration**.

6 From the table, select the Logical Video bundle type with the BundleWizard template type.

7 To add a Wizard Template, browse to the "`SampleMetadataWizardTabAction.xml`" file.

8 Click **Save**.

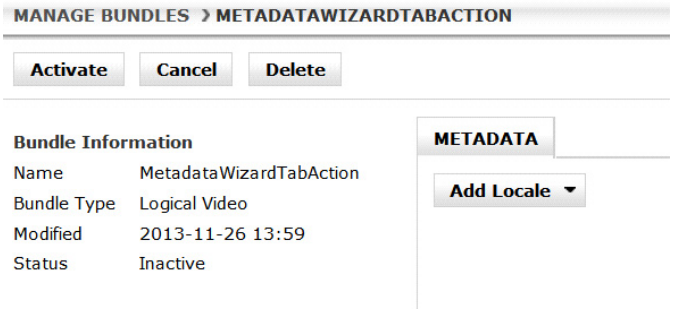


9 Navigate to **Producer > Manage Bundles > Add New > Logical Video**.

a Enter a name.

b Click **Create & Edit**.

Note The new bundle must contain the bundle information and "METADATA" tab.



Subtitle Wizard Tab

Plug-in Architecture

To view the plug-in architecture, see the attached Plug-in Architecture zip file.

Description

The subtitle wizard tab allows users to configure custom subtitle settings for specific bundles.

To view the methods this interfaces specifies, open the Content Manager folder in “vms_docs-[releasenumber]-[buildnumber]” and select “index.html”.

Example

The following is a sample Subtitle Wizard Tab:

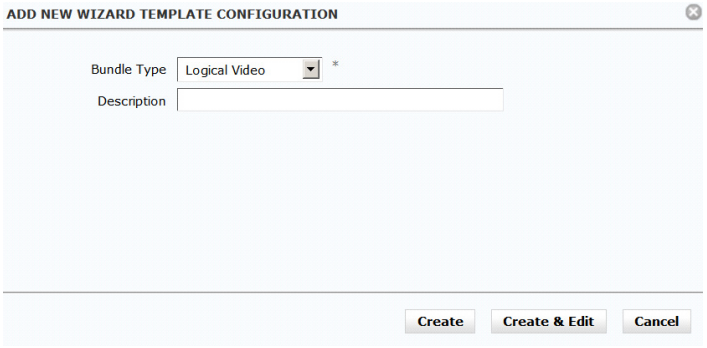
```
package com.cisco.tab.identifier;

import java.util.ArrayList;
import java.util.List;

import org.jboss.seam.ScopeType;
import org.jboss.seam.annotations.Name;
import org.jboss.seam.annotations.Scope;

import com.extend.opencase.cm.action.component.template.CompoundFormElements;
import com.extend.opencase.cm.model.component.AbstractComponent;
import com.extend.opencase.cm.model.component.ComponentType;
import com.extend.opencase.cm.model.component.bundle.BundleItem;
import com.extend.opencase.cm.model.component.identifier.AbstractIdentifier;
import com.extend.opencase.cm.webservice.fault.ComponentParameterFault;
import com.extend.opencase.cm.webservice.fault.ComponentWebserviceFault;
import com.extend.opencase.pt.action.wizard.tab.common.AbstractWizardTabSearchActions;
import com.extend.opencase.pt.action.wizard.tab.common.ComponentWebServiceSearchResult;

@Name("identifierWizardTabAction")
@Scope(ScopeType.PAGE)
public class IdentifierWizardTabAction extends AbstractWizardTabSearchActions {

          private static final String IDENTIFIERS_FOLDER = "Identifiers";

          private static final long serialVersionUID = 1L;

          private List<ComponentType> availableComponentTypes;

          public IdentifierWizardTabAction() {
```

```
                super();
        }

        @Override
        protected void init() {
                super.init();

                initComponentsMap(AbstractIdentifier.class);
                initTree();

                selectFirstFolder();
        }

        @Override
        protected void initComponentsMap(Class<?> componentClass) {
                super.initComponentsMap(componentClass);

                initAvailableComponentTypes();
        }

        @Override
        public void save() {
                super.save();
                refreshTreeAfterUpdate();
        }

        @Override
        public void delete() {
                super.delete();
                refreshTreeAfterDelete(AbstractIdentifier.class, "mainAreaDiv:identifierTab");
        }

        @Override
        public CompoundFormElements getCompoundFormElements() {
                refreshTabPanel("mainAreaDiv:identifierTab");
                return super.getCompoundFormElements();
        }

        public void addSelected() {
                addSelected(AbstractIdentifier.class, "mainAreaDiv:identifierTab");
        }

        public void add(ComponentType componentType) {
                AbstractIdentifier newIdentifier = (AbstractIdentifier) componentType.getInstanceClass();
                newIdentifier.setName(componentType.getName());

                try {
                        AbstractIdentifier createdIdentifier = (AbstractIdentifier)
getComponentWebService().createComponent(newIdentifier);

                        getComponentWebService().addComponentToBundle(createdIdentifier.getComponentPk(),
getCurrentBundle().getComponentPk(),
                                        folder.getBundleItemPk());

                        initComponentsMap(AbstractIdentifier.class);
                        initTree();
                        selectFolder(folder);
                } catch(ComponentParameterFault componentParameterFault) {
                        addFailedMessage(componentParameterFault, "Failed to create new identifier component.");
                } catch(ComponentWebserviceFault componentWebserviceFault) {
                        addFailedMessage(componentWebserviceFault, "Failed to create new identifier component.");
                }

}
```

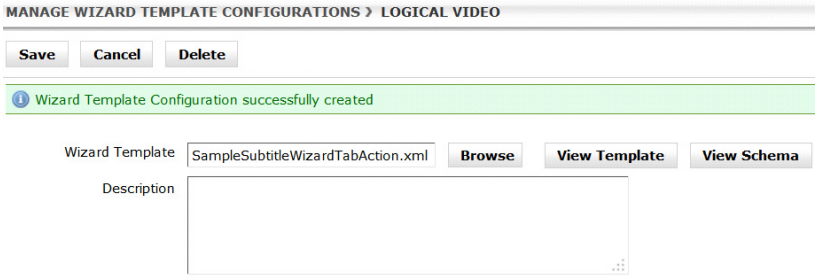
Building the Plug-in

Follow the steps below to create a custom Subtitle Wizard tab in 5.1.x versions of VMS:

1. Navigate to **Producer > Setup > Wizard Template Configurations**.
2. Click **Add New**.
3. Select Logical Video as the bundle type on the "Add New Wizard Template Configuration" window.
4. Click **Create & Edit.**

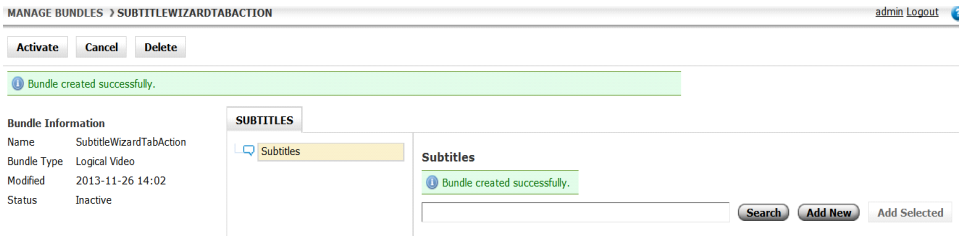


5. To add a Wizard Template, browse to the "`SampleSubtitleWizardTabAction.xml`" file.
6. Click **Save**.



7. Navigate to **Producer > Manage Bundles > Add New > Logical Video**.
   a. Enter a name.
   b. Click **Create & Edit**.

Note The new bundle must contain the bundle information and "SUBTITLES" tab.



Identifier Wizard Tab

Plug-in Architecture

To view the plug-in architecture, see the attached Plug-in Architecture zip file.

Description

The custom Identifier wizard tab will reflect chosen identifier components, including: TMS, ISAN, or EIDR. For more options, see the configuration tree node.

Example

The following is a sample Identifier Wizard Tab:

```
package com.cisco.tab.identifier;

import java.util.ArrayList;
import java.util.List;
ISA
import org.jboss.seam.ScopeType;
import org.jboss.seam.annotations.Name;
import org.jboss.seam.annotations.Scope;

import com.extend.opencase.cm.action.component.template.CompoundFormElements;
import com.extend.opencase.cm.model.component.AbstractComponent;
import com.extend.opencase.cm.model.component.ComponentType;
import com.extend.opencase.cm.model.component.bundle.BundleItem;
import com.extend.opencase.cm.model.component.identifier.AbstractIdentifier;
import com.extend.opencase.cm.webservice.fault.ComponentParameterFault;
import com.extend.opencase.cm.webservice.fault.ComponentWebserviceFault;
import com.extend.opencase.pt.action.wizard.tab.common.AbstractWizardTabSearchActions;
import com.extend.opencase.pt.action.wizard.tab.common.ComponentWebServiceSearchResult;

@Name("identifierWizardTabAction")
@Scope(ScopeType.PAGE)
public class IdentifierWizardTabAction extends AbstractWizardTabSearchActions {

          private static final String IDENTIFIERS_FOLDER = "Identifiers";

          private static final long serialVersionUID = 1L;

          private List<ComponentType> availableComponentTypes;

          public IdentifierWizardTabAction() {
                    super();
          }

          @Override
          protected void init() {
                    super.init();
```

```
                initComponentsMap(AbstractIdentifier.class);
                initTree();

                selectFirstFolder();
        }

        @Override
        protected void initComponentsMap(Class<?> componentClass) {
                super.initComponentsMap(componentClass);

                initAvailableComponentTypes();
        }

        @Override
        public void save() {
                super.save();
                refreshTreeAfterUpdate();
        }

        @Override
        public void delete() {
                super.delete();
                refreshTreeAfterDelete(AbstractIdentifier.class, "mainAreaDiv:identifierTab");
        }

        @Override
        public CompoundFormElements getCompoundFormElements() {
                refreshTabPanel("mainAreaDiv:identifierTab");
                return super.getCompoundFormElements();
        }

        public void addSelected() {
                addSelected(AbstractIdentifier.class, "mainAreaDiv:identifierTab");
        }

        public void add(ComponentType componentType) {
                AbstractIdentifier newIdentifier = (AbstractIdentifier) componentType.getInstanceClass();
                newIdentifier.setName(componentType.getName());

                try {
                        AbstractIdentifier createdIdentifier = (AbstractIdentifier)
getComponentWebService().createComponent(newIdentifier);

                        getComponentWebService().addComponentToBundle(createdIdentifier.getComponentPk(),
getCurrentBundle().getComponentPk(),
                                        folder.getBundleItemPk());

                        initComponentsMap(AbstractIdentifier.class);
                        initTree();
                        selectFolder(folder);
                } catch(ComponentParameterFault componentParameterFault) {
                        addFailedMessage(componentParameterFault, "Failed to create new identifier component.");
                } catch(ComponentWebserviceFault componentWebserviceFault) {
                        addFailedMessage(componentWebserviceFault, "Failed to create new identifier component.");
                }

}
```

Building the Plug-in

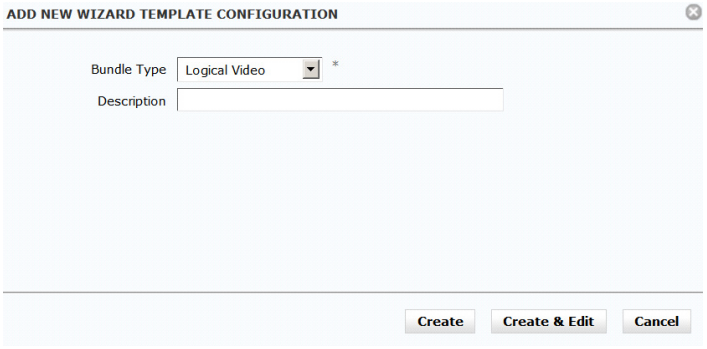
Follow the steps below to build a custom Identifier Wizard tab using a 5.1.x version of VMS:

1 Navigate to **Producer > Setup > Wizard Template Configurations**.

2 Click **Add New**.

3 Select Logical Video as the bundle type on the "Add New Wizard Template Configuration" window.

4 Click **Create & Edit.**



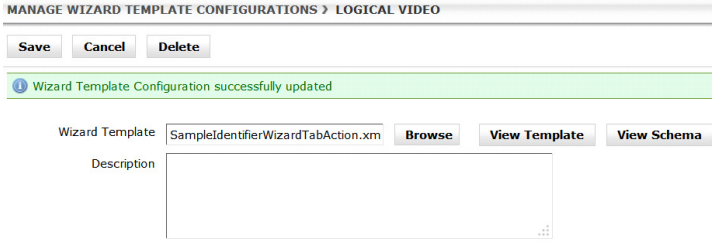
Follow the steps below to build a custom Identifier Wizard tab using a 5.5 version of VMS

1 Navigate to **Producer > Setup > Wizard Template Configuration**.

2 From the table, select the Logical Video bundle type with the "BundleWizard"

3 To add a Wizard Template, browse to the "`SampleIdentifierWizardTabAction.xml`" file.

4 Click **Save.**

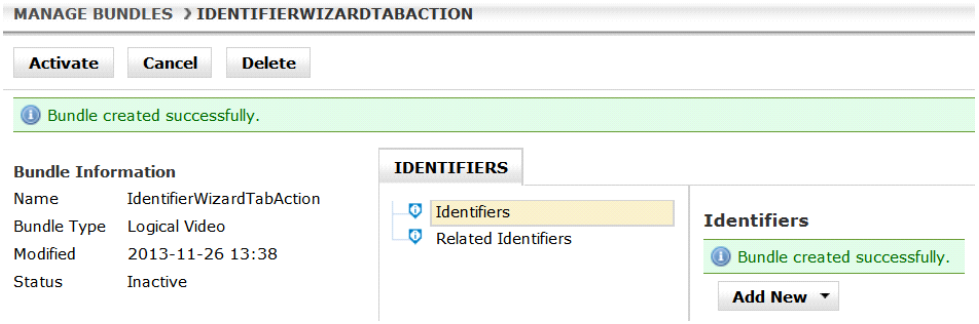


5 Navigate to **Producer > Manage Bundles > Add New > Logical Video**.

   a Enter a name.

   b Click **Create & Edit**.

Note The new bundle must contain the bundle information and "IDENTIFIERS" tab.



Triggering the Sample Plug-in

Metadata Source Plug-in

Plug-in Architecture

To view the plug-in architecture, see the attached Plug-in Architecture zip file.

Description

The Metadata Source plug-in is used to search for metadata and images from 3rd party metadata providers. A movie or show title is used as the search term. The core VMS product provides support for three build-in search providers: TMS, TMDB and EIDR. Support for any other provider can be implemented as a separate plug-in and deployed into VMS. A Sample Metadata Search Plug-in is included in implementation, which duplicates build-in search functionality for the TMS provider but as an SDK plug-in. This sample can modified to fit new search provider requirements.

The following files are included with the sample TMS plug-in:

`SampleAbstractMetadataSourcePlugin.java`

This abstract class implements the most common `MetadataSourcePlugin` interface methods (such as HTTP connect, access to the plug-in configuration settings, etc). This class should be reused for other metadata search plug-ins with similar functionality.

`SampleDefaultProviderFieldFactory.java`

This file implements `ProviderFieldFactory` methods for data conversion between search provider data format and VMS data format.

`SampleMetadataSourcePlugin.java`

This file extends the `SampleAbstractMetadataSourcePlugin` class and implements major interface methods, including: `searchMetadata`, `fetchDetails`, and `searchImages`.

`SampleTMSProviderFieldFactory.java`

This file extends `SampleDefaultProviderFieldFactory` and adds provider specific data file formats (for example, specific date patterns).

To view the methods this interfaces specifies, open the Content Manager folder in "vms_docs-[releasenumber]-[buildnumber]" and select "index.html".

Metadata Plug-in Interface Methods to Implement

```
/**
 * Method for initializing plugin configuration before calling search methods.
 *
 * @param config Configuration for the plugin
 */
void initialize(MetadataSourceConfig config);

/**
 * Gets set of fields for update by locale field mapping
 *
 * @param locale locale
 * @return set of fields for update
 * @throws MetadataSourcePluginException on error
 */
Set<String> getFilteredFieldsByLocal(String locale) throws MetadataSourcePluginException;

/**
 * Gets the configuration of field mappings
 *
 * @return The configuration of field mappings
 * @throws MetadataSourcePluginException on error
 */
MappingConfig getFieldMapping() throws MetadataSourcePluginException;

/**
 * Method for searching Metadata.
        *
        * @param searchTerm Encoded search string
        * @param params Other parameters
        * @param startPage Page number starting from 1, can be null for first page
        * @param maxResultSize For future use, custom page size is not supported
        * @return MetadataSearchResult
        * @throws MetadataSourcePluginException the metadata source plugin exception
        * @throws IOException Signals that an I/O exception has occurred.
*/
MetadataSearchResult searchMetadata(String searchTerm, Map<String, String> params, Integer
startPage, Integer maxResultSize)
       throws MetadataSourcePluginException, IOException;

/**
* Method for searching Images.
 *
 * @param searchTerm Encoded search string, can be null
 * @param params Other parameters
 * @param startPage Page number starting from 1, can be null for first page
 * @param maxResultSize For future use, custom page size is not supported
 * @return A ImageSearchResult
```

```
 * @throws MetadataSourcePluginException the metadata source plugin exception
 * @throws IOException Signals that an I/O exception has occurred.
 */
ImageSearchResult searchImages(String searchTerm, Map<String, String> params, Integer
startPage, Integer maxResultSize)
                      throws MetadataSourcePluginException, IOException;

/**
 * Method for searching Images for a particular movie
 *
 * @param identifier the plugin movie identifier
 * @param params the additional parameters
 * @return the image search result
 * @throws MetadataSourcePluginException the metadata source plugin exception
 * @throws IOException Signals that an I/O exception has occurred.
*/
ImageSearchResult searchImages(String identifier, Map<String, String> params) throws
MetadataSourcePluginException, IOException
;

/**
* Method to get Metadata details.
 *
* @param identifier Metadata id
* @param locale the locale (if null or if locale fieldmap is not configured, no fields are
returned)
 * @return Metadata details
 * @throws MetadataSourcePluginException the metadata source plugin exception
* @throws IOException Signals that an I/O exception has occurred.
*/
MetadataDetails fetchDetails(String identifier, String locale) throws
MetadataSourcePluginException, IOException;

/**
* Method for indicating whether this implementation supports Metadata search, may return false
for image only repository.
 *
* @return the boolean
* @throws MetadataSourcePluginException the metadata source plugin exception
 */
Boolean providesMetadataSearch() throws MetadataSourcePluginException;

/**
 * Method for indicating whether this implementation supports Image search, may return false for
metadata only repository e.g. EIDR
 *
* @return the boolean
* @throws MetadataSourcePluginException the metadata source plugin exception
*/
Boolean providesImageSearch() throws MetadataSourcePluginException;

/**
 * Supported page sizes (number of results per page) for pagination.
```

```
 *
* @return the list of supported page sizes
 */
List<Integer> supportedPageSizes();

/**
* Gets the customized provider field factory that is going to implemented by the developer of
the plug-in
*
* @return The instance of implementation of {@code ProviderFieldFactory}
 */
ProviderFieldFactory getProviderFieldFactory();

/**
* Method returns list of configurations required for plugin.
*
 * @return list of configurations required for plugin
 */
List<MetadataConfigurationKey> getConfigurationsKeys();
```

Metadata Source Plug-in Configuration Settings

Each plug-in contains a set of configuration settings that can be added or updated from the "Manage Metadata Source Plugins" page in the UI. To add support for plug-in settings, the following code can be used (see `SampleMetadataSourcePlugin.java`):

```
private static final String URL_KEY = "url";
private static final String API_KEY_KEY = "apiKey";
private static final String IMAGE_BASE_URL_KEY = "imagesBaseUrl";

private static final List<MetadataConfigurationKey> CONFIG_FIELDS = Arrays.asList(
       new MetadataConfigurationKey("url", true, false),
       new MetadataConfigurationKey("apiKey", true, false),
new MetadataConfigurationKey("imagesBaseUrl", false, false));
```

The code above will expose three additional configuration settings: url, apiKey, and imagesBaseUrl in the UI.

The constructor for the MetadataConfigurationKey class is defined as follows:

```
/**
* Constructor.
 *
* @param name name of the configuration property
* @param isRequired parameter to specify that this configuration is required for plugin
* @param isProtected parameter to specify that this configuration is should be encrypted
 */
public MetadataConfigurationKey(String name, boolean isRequired, boolean isProtected)
```

To retrieve an administrator entered value on the UI from the plug-in source code, use the following syntax:

```
getProperty(URL_KEY)
```

Field Mapping

Field mapping exists in an XML file that is uploaded to the "Manage Metadata Source Plugins" page. The file provides mapping between 3rd party provider data fields and VMS component fields.

Mappings are included in "component" tags and the "type" attribute defines a VMS component type.

```
<component type="Metadata" folder="Metadata Folder">
```

The mapping for a simple VMS field would look like the following:

```
<field name="summary" type="string" provider_field_name="shortDescription" />
```

In the example above, "name" is a VMS metadata field while "provider_field_name" is a filed name/expression that instructs plug-ins how and where to extract this field from a provider feed. Specifically, TMS returns data in a JSON format and the "short description" is described in this format as follows:

```
{         "hitCount": 12702,
          "hits": [{
                    "program": {
                              "tmsId": "SH015265930000",
                              "title": "Dreams",
                              "titleLang": "en",
                    "shortDescription": "A film by Allan King.",
                           "longDescription": "A film by Allan King.",
                    "descriptionLang": "en",
                    "subType": "Special",
                    "genres": ["Special","Drama"],
                             "origAirDate": "2012-02-03",
                             "preferredImage": {"uri":
                              "tvbanners/generic/generic_tvbanners_v5.png"},
                              "rootId": "9060283",
                              "entityType": "Show"
}   }
```

A "type" attribute expresses the type of data format "provider_field_name" is represented as in feeds. This allows users to cast data correctly.

Mapping for a complex field (e.g. genre) is defined with the <commonEntity> tag and will look like the following:

```
<commonEntity type="Genre" fieldName="genres" collection="true">
             <field name="genreName" type="string" provider_field_name="genres[*]"
enumerated="true" separator="," />
</commonEntity>
```

In the example above, "provider_field_name" include a full JSONPath expression which fetches all genres. The attribute "enumerated" means that all genres are concatenated into a long string and "separator" means that individual genres within the string are separated by a ",".

All mappings that are uploaded to Producer from the UI will be validated against the following schema:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<xs:schema version="1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema">
	<xs:simpleType name="stringType">
		<xs:restriction base="xs:string" />
	</xs:simpleType>

	<xs:simpleType name="intType">
		<xs:restriction base="xs:int" />
	</xs:simpleType>

	<xs:simpleType name="booleanType">
		<xs:restriction base="xs:boolean" />
	</xs:simpleType>

	<xs:complexType name="fieldType">
		<xs:attributeGroup ref="fieldattr" />
	</xs:complexType>

	<xs:complexType name="commonEntityType">
		<xs:sequence>
			<xs:element name="field" type="fieldType" minOccurs="1"
maxOccurs="unbounded" />
		</xs:sequence>
		<xs:attribute ref="type" use="required" />
		<xs:attribute ref="collection" />
	 <xs:attribute ref="fieldName" use="required"/>
	</xs:complexType>

	<xs:complexType name="customAttributeType">
		<xs:all>
			<xs:element name="field" type="fieldType" minOccurs="1" maxOccurs="1" />
		</xs:all>
	 <xs:attribute ref="name" />
    </xs:complexType>

    <xs:complexType name="componentType">
		<xs:sequence>
			<xs:element name="field" type="fieldType" minOccurs="0"
maxOccurs="unbounded" />
			<xs:element name="commonEntity" type="commonEntityType" minOccurs="0"
maxOccurs="unbounded" />
			<xs:element name="customAttribute" type="customAttributeType"
minOccurs="0" maxOccurs="unbounded" />
		</xs:sequence>
		<xs:attribute ref="type" use="required" />
		<xs:attribute ref="folder" use="required" />
	</xs:complexType>

	<xs:complexType name="mappingType">
		<xs:sequence>
```

```
                        <xs:element name="component" type="componentType" minOccurs="1"
maxOccurs="unbounded" />
                </xs:sequence>
        </xs:complexType>

        <xs:attribute name="type" type="stringType" />
        <xs:attribute name="folder" type="stringType" />
        <xs:attribute name="fieldName" type="stringType" />
        <xs:attribute name="name" type="stringType" />
        <xs:attribute name="provider_field_name" type="stringType" />
        <xs:attribute name="value" type="stringType" />
        <xs:attribute name="enumerated" type="booleanType" default="false" />
        <xs:attribute name="separator" type="stringType" fixed="," />
        <xs:attribute name="collection" type="booleanType" default="false" />
        <xs:attribute name="dateFormat" type="stringType" />

        <xs:attributeGroup name="fieldattr">
                <xs:attribute ref="name" use="required" />
                <xs:attribute ref="provider_field_name"/>
                <xs:attribute ref="type" use="required" />
                <xs:attribute ref="value"/>
                <xs:attribute ref="enumerated" />
                <xs:attribute ref="separator" />
                <xs:attribute ref="dateFormat" />
        </xs:attributeGroup>

        <xs:element name="mapping" type="mappingType"/>
</xs:schema>
```

Examples

The following is a sample of the TMS mapping:

```
<mapping>
               <component type="Metadata" folder="Metadata Folder">
                              <field name="episodeName" type="string" provider_field_name="episodeTitle" />
                              <field name="episodeNumber" type="string" provider_field_name="episodeNum" />
                              <field name="numberOfEpisodes" type="int" provider_field_name="totalEpisodes" />
                              <field name="seasonNumber" type="string" provider_field_name="seasonNum" />
                              <field name="titleShort" type="string" provider_field_name="title" />
                              <field name="titleMedium" type="string" provider_field_name="title" />
                              <field name="titleLong" type="string" provider_field_name="title" />
                              <field name="titleSortable" type="string" provider_field_name="title" />
                              <field name="summaryShort" type="string" provider_field_name="shortDescription" />
                              <field name="summaryMedium" type="string" provider_field_name="shortDescription" />
                              <field name="summaryLong" type="string" provider_field_name="longDescription" />
                              <field name="summary" type="string" provider_field_name="shortDescription" />
                              <field name="duration" type="string" provider_field_name="runTime" />
                            <field name="releaseDate" type="datetime" provider_field_name="releaseYear" dateFormat="yyyy"
/>
                            <field name="firstAirDate" type="datetime" provider_field_name="origAirDate" dateFormat="yyyy-
MM-dd" />
                                                        <field name="keywords" type="string"
provider_field_name="keywords.Subject" enumerated="true" separator="," />

                              <commonEntity type="Talent" fieldName="castMembers" collection="true">
                                                <field name="person.firstName,person.lastName" type="string"
provider_field_name="cast[*].name" enumerated="true" separator="," />
                                                <field name="person.displayName" type="string"
provider_field_name="cast[*].name" enumerated="true" separator="," />
                                                <field name="talentRole.roleName" type="string" value="castMember" />
```

```
                                </commonEntity>

                                <commonEntity type="Talent" fieldName="producers" collection="true">
                                        <field name="person.firstName,person.lastName" type="string"
provider_field_name="crew[?(@.role == 'Producer')].name" enumerated="true" separator="," />
                                        <field name="person.displayName" type="string"
provider_field_name="crew[?(@.role == 'Producer')].name" enumerated="true" separator="," />
                                        <field name="talentRole.roleName" type="string" value="producer" />
                                </commonEntity>

                                <commonEntity type="Talent" fieldName="directors" collection="true">
                                        <field name="person.firstName,person.lastName" type="string"
provider_field_name="crew[?(@.role == 'Director')].name" enumerated="true" separator="," />
                                        <field name="person.displayName" type="string"
provider_field_name="crew[?(@.role == 'Director')].name" enumerated="true" separator="," />
                                        <field name="talentRole.roleName" type="string" value="director" />
                                </commonEntity>

                                <commonEntity type="Talent" fieldName="writers" collection="true">
                                        <field name="person.firstName,person.lastName" type="string"
provider_field_name="crew[?(@.role == 'Writer')].name" enumerated="true" separator="," />
                                        <field name="person.displayName" type="string"
provider_field_name="crew[?(@.role == 'Writer')].name" enumerated="true" separator="," />
                                        <field name="talentRole.roleName" type="string" value="writer" />
                                </commonEntity>

                                <commonEntity type="Genre" fieldName="genres" collection="true">
                                        <field name="genreName" type="string" provider_field_name="genres[*]"
enumerated="true" separator="," />
                                </commonEntity>

                                <commonEntity type="Advisory" fieldName="advisories" collection="true">
                                        <field name="advisoryName" type="string" provider_field_name="advisories[*]"
enumerated="true" separator="," />
                                </commonEntity>

                                <commonEntity type="Rating" fieldName="ratings" collection="true">
                                        <field name="ratingName" type="string" provider_field_name="ratings[?(@.body
== 'Motion Picture Association of America')].code" enumerated="true" separator="," />
                                        <field name="ratingBody" type="string" provider_field_name="ratings[?(@.body
== 'Motion Picture Association of America')].body" enumerated="true" separator="," />
                                </commonEntity>
                </component>
</mapping>
```

The following is a sample Metadata Source plug-in:

```
package com.cisco.vms.plugins.metadata.augmentation.tms;

import java.io.IOException;
import java.net.URLEncoder;
import java.text.MessageFormat;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
import java.util.Locale;
import java.util.Map;
import java.util.Map.Entry;
import java.util.Properties;

import com.extend.opencase.pt.plugin.MetadataSourcePluginException;
import com.extend.opencase.pt.plugin.ProviderFieldFactory;
import com.extend.opencase.pt.session.manager.augmentation.config.MappingConfigUtil;

import net.minidev.json.JSONObject;

import org.apache.commons.lang.LocaleUtils;
import org.apache.commons.lang.NotImplementedException;
import org.apache.commons.lang.StringUtils;
import org.slf4j.Logger;
```

```
import org.slf4j.LoggerFactory;

import com.extend.opencase.occ.client.util.TimeUtil;
import com.extend.opencase.pt.model.metadatasource.ImageDetails;
import com.extend.opencase.pt.model.metadatasource.ImageSearchResult;
import com.extend.opencase.pt.model.metadatasource.MetadataConfigurationKey;
import com.extend.opencase.pt.model.metadatasource.MetadataDetails;
import com.extend.opencase.pt.model.metadatasource.MetadataSearchResult;
import com.extend.opencase.pt.model.metadatasource.MetadataSourceConfigFieldsEnum;
import com.extend.opencase.pt.model.metadatasource.MetadataSourcePluginEntityTypeEnum;
import com.extend.opencase.pt.model.metadatasource.mapping.MappingConfigField;
import com.jayway.jsonpath.InvalidPathException;
import com.jayway.jsonpath.JsonPath;

/**
 * This is the implementation class for the TMS Metadata source plugin.
 */
public class SampleMetadataSourcePlugin extends SampleAbstractMetadataSourcePlugin {

	private static final long serialVersionUID = 4427802192503026877L;
	private static final Logger LOG = LoggerFactory.getLogger(SampleMetadataSourcePlugin.class);
	private static final String ID_CANT_BE_NULL = "ID can't be null";
	private static final String LOCALE_CANT_BE_NULL = "Locale can't be null";
	private static final String SEARCH_TERM_CANT_BE_EMPTY = "Search term cannot be empty.";
	private static final String MAX_RESULT_LIMIT = "maxResultSize for a TMS programs search is out of 1-50 range
limit.";
	private static final String FORWARD_SLASH = "/";
	private static final String DURATION_FIELD = "runTime";

	private static final String ENTITY_TYPE_MOVIE = "Movie";
	private static final String ENTITY_TYPE_SERIES = "Series";
	private static final String ENTITY_TYPE_EPISODE = "Episode";
	private static final String ENTITY_TYPE_SHOW = "Show";
	private static final String ENTITY_TYPE_SPORTS = "Sports";

	private static final List<Integer> SUPPORTED_PAGE_SIZES = Arrays.asList(10, 30, 50);
	private static final SampleTMSProviderFieldFactory TMS_PROVIDER_FIELD_FACTORY = new
SampleTMSProviderFieldFactory();
	private static final List<MetadataConfigurationKey> CONFIG_FIELDS = Arrays.asList(
			new MetadataConfigurationKey(MetadataSourceConfigFieldsEnum.url, true, false),
			new MetadataConfigurationKey(MetadataSourceConfigFieldsEnum.apiKey, true, false),
			new MetadataConfigurationKey(MetadataSourceConfigFieldsEnum.imagesBaseUrl, false, false));

	/**
	 * Constructor
	 */
	public SampleMetadataSourcePlugin() {
		super();
	}

	/**
	 * {@inheritDoc}
	 */
	@Override
	public MetadataSearchResult searchMetadata(String searchTerm, Map<String, String> params, Integer startPage,
Integer maxResultSize)
			throws MetadataSourcePluginException, IOException {
		LOG.debug("Begin searchMetadata(): searchTerm=[{}], params=[{}], startPage=[{}], maxResultSize=[{}].",
new Object[] { searchTerm,
				params, startPage, maxResultSize });

		Long time = System.currentTimeMillis();

		if(StringUtils.isBlank(searchTerm)) {
			LOG.error(SEARCH_TERM_CANT_BE_EMPTY);
			throw new MetadataSourcePluginException(SEARCH_TERM_CANT_BE_EMPTY);
		}

		// get config and handle searchTerm parameter
		Properties configProps = getConfig().getBaseConfig();
		StringBuilder url = new StringBuilder(MessageFormat.format("{0}/programs/search?api_key={1}&q={2}",
				configProps.getProperty(MetadataSourceConfigFieldsEnum.url.toString()).trim(),
```

```
                                        configProps.getProperty(MetadataSourceConfigFieldsEnum.apiKey.toString()).trim(),
URLEncoder.encode(searchTerm.trim(), "UTF-8")));

                    // handle maxResultSize parameter
                    if(maxResultSize != null) {
                              // validate TMS max results which is between 1 and 50
                              if(maxResultSize < 0 || maxResultSize > 50) {
                                        LOG.error(MAX_RESULT_LIMIT);
                                        throw new MetadataSourcePluginException(MAX_RESULT_LIMIT);
                              }

                              url.append("&limit=").append(maxResultSize.toString());
                    }

                    // handle startPage parameter
                    if(startPage != null) {
                              // default page size is 10 for TMS
                              int pageLimit = maxResultSize == null ? 10 : maxResultSize;
                              Integer offset = (startPage - 1) * pageLimit;
                              if(offset > 0) {
                                        url.append("&offset=").append(offset.toString());
                              }
                    }

                    if(params != null && !params.isEmpty()) {
                              // handle locale parameter
                              String locale = params.get(MetadataSourceConfigFieldsEnum.locale.toString());
                              if(StringUtils.isNotBlank(locale)) {
                                        Locale localeObj = LocaleUtils.toLocale(locale);
                                        String language = localeObj.getLanguage();
                                        if(StringUtils.isNotBlank(language)) {

url.append("&titleLang=").append(language.trim()).append("&descriptionLang=").append(language.trim());
                                        }
                              }
                    }

                    // invoke TMS API service for program search
                    String searchResponse = fetchHttpBody(url.toString());

                    LOG.debug("Retrieving metadata from TMS took {}", (System.currentTimeMillis() - time) + TimeUtil.MSEC);
                    LOG.debug("Search response from TMS ProgramSearch: [{}]", searchResponse);

                    time = System.currentTimeMillis();
                    MetadataSearchResult result = prepareSearchResultForProgramSearch(searchResponse);

                    LOG.debug("Parsing TMS search result took {}. MetadataSearchResult = [{}]", (System.currentTimeMillis()
- time) + TimeUtil.MSEC, result);

                    return result;
          }

          /**
           * Gets the image base url.
           *
           * @param configProps the plugin configuration properties
           * @return the image base url
           */
          private String getImageBaseUrl(Properties configProps) {
                    String baseImagesPath =
configProps.getProperty(MetadataSourceConfigFieldsEnum.imagesBaseUrl.toString());
                    if(StringUtils.isNotEmpty(baseImagesPath)) {
                              if(!baseImagesPath.endsWith(FORWARD_SLASH)) {
                                        baseImagesPath += FORWARD_SLASH;
                              }
                              return baseImagesPath;
                    }

                    return null;
          }

          /**
```

```
         * Prepare search result for metadata search.
         *
         * @param searchResponse string containing JSON response from TMS API
         * @return MetadataSearchResult
         * @throws MetadataSourcePluginException the metadata source plugin exception
         */
        private MetadataSearchResult prepareSearchResultForProgramSearch(final String searchResponse) throws
MetadataSourcePluginException {
                // populate response with metadata information
                MetadataSearchResult metadataSearchResult = new MetadataSearchResult();

                try {
                        List<JSONObject> resultList = JsonPath.read(searchResponse, "$.hits[*].program");
                        Properties configProps = getConfig().getBaseConfig();
                        for(JSONObject resultMap : resultList) {
                                String jsonString = resultMap.toJSONString();
                                MetadataDetails metadataDetails = new MetadataDetails();

                                // skip Sports type of program
                                try {
                                        String entityType = JsonPath.read(jsonString, "entityType");
                                        if(StringUtils.isNotEmpty(entityType)) {
                                                switch(entityType) {
                                                        case ENTITY_TYPE_MOVIE:

metadataDetails.setEntityType(MetadataSourcePluginEntityTypeEnum.MOVIE);
                                                                break;
                                                        case ENTITY_TYPE_SERIES:

metadataDetails.setEntityType(MetadataSourcePluginEntityTypeEnum.SERIES);
                                                                break;
                                                        case ENTITY_TYPE_SHOW:

metadataDetails.setEntityType(MetadataSourcePluginEntityTypeEnum.SHOW);
                                                                break;
                                                        case ENTITY_TYPE_EPISODE:

metadataDetails.setEntityType(MetadataSourcePluginEntityTypeEnum.EPISODE);
                                                                break;
                                                        case ENTITY_TYPE_SPORTS:
                                                                // skip sports
                                                                continue;
                                                        default:
                                                                // set nothing - keep default UNKNOWN
                                                                break;
                                                }
                                        }
                                } catch(InvalidPathException ex) {
                                        LOG.debug("Couldn't get entityType from TMS response.", ex);
                                }

                                // fetch TMS ID - need it to retrieve movie details later on
                                String tmsId = JsonPath.read(jsonString, "tmsId");
                                metadataDetails.setIdentifier(tmsId);

                                // fetch image relative url and build absolute path
                                try {
                                        String imageUrl = JsonPath.read(jsonString, "preferredImage.uri");
                                        String baseImagesPath = getImageBaseUrl(configProps);
                                        if(StringUtils.isNotEmpty(baseImagesPath)) {
                                                metadataDetails.setImageUrl(baseImagesPath + imageUrl);
                                        } else {
                                                metadataDetails.setImageUrl(imageUrl);
                                        }
                                } catch(InvalidPathException ex) {
                                        LOG.debug("Couldn't get preferredImage.uri from TMS", ex);
                                }

                                // collect plugin data in format: tmsKey=tmsRawValue
                                Map<String, Object> values = metadataDetails.getRawValues();
                                for(Entry<String, MappingConfigField> entry : getFieldMap().entrySet()) {
                                        String tmsKey = entry.getValue().getProviderFieldName();
```

```
                                        if(StringUtils.isNotEmpty(tmsKey)) {
                                                try {
                                                        values.put(tmsKey, JsonPath.read(jsonString, tmsKey));
                                                } catch(InvalidPathException ex) {
                                                        LOG.debug("Couldn't read from TMS a value for key:
[{}]", tmsKey);
                                                }
                                        }
                                }

                                if(!values.isEmpty()) {
                                        metadataSearchResult.getResults().add(metadataDetails);
                                }
                        }

                        // populate response with additional info
                        Map<String, Object> additionalInfo = new HashMap<String, Object>();
                        additionalInfo.put("total_results", JsonPath.read(searchResponse, "$.hitCount").toString());
                        metadataSearchResult.setAdditionalInfo(additionalInfo);
                } catch(InvalidPathException ex) {
                        throw new MetadataSourcePluginException(ex);
                }

                return metadataSearchResult;
        }

        /**
         * {@inheritDoc}
         */
        @Override
        public ImageSearchResult searchImages(String searchTerm, Map<String, String> params, Integer startPage, Integer
maxResultSize)
                        throws MetadataSourcePluginException, IOException {
                LOG.debug("Begin searchImages(String searchTerm, Map<String, String> params, Integer startPage, Integer
maxResultSize).");

                throw new NotImplementedException(
                                "Method searchImages(String searchTerm, Map<String, String> params, Integer
startPage, Integer maxResultSize) is not implemented for TMS plugin.");
        }

        @Override
        public ImageSearchResult searchImages(String identifier, Map<String, String> params) throws
MetadataSourcePluginException, IOException {
                LOG.debug("Begin TMS searchImages: identifier=[{}], params=[{}].", identifier, params);

                Long time = System.currentTimeMillis();

                if(StringUtils.isEmpty(identifier)) {
                        LOG.error("TMS movie ID is required to retrieve images.");
                        throw new MetadataSourcePluginException("TMS movie ID is required to retrieve images.");
                } else {
                        ImageSearchResult isr = new ImageSearchResult();
                        ArrayList<ImageDetails> imageDetailsList = new ArrayList<ImageDetails>();
                        // image search with id parameter
                        ImageDetails imageDetails = new ImageDetails();
                        imageDetails.setImages(getImages(identifier, params));
                        // populate response with image information
                        imageDetailsList.add(imageDetails);
                        isr.setResults(imageDetailsList);

                        LOG.debug("End searchImages() {}. ImageSearchResult = {}", (System.currentTimeMillis() - time)
+ TimeUtil.MSEC, isr);

                        return isr;
                }
        }

        /**
         * This method gets the images based on id
         *
```

```
	 * @param identifier the identifier
	 * @param params the image search parameters passed to the search
	 * @return the images
	 * @throws MetadataSourcePluginException the metadata source plugin exception
	 * @throws IOException Signals that an I/O exception has occurred.
	 */
	private List<Map<String, Object>> getImages(String identifier, Map<String, String> params) throws
MetadataSourcePluginException, IOException {

		LOG.debug("Invoked getImages() with identifier=[{}], params=[{}].", identifier, params);

		if(StringUtils.isBlank(identifier)) {
			LOG.error(ID_CANT_BE_NULL);
			throw new MetadataSourcePluginException(ID_CANT_BE_NULL);
		}

		Properties configProps = getConfig().getBaseConfig();
		String searchResponse = fetchHttpBody(MessageFormat.format("{0}/programs/{1}/images?api_key={2}",
				configProps.getProperty(MetadataSourceConfigFieldsEnum.url.toString()).trim(),
identifier.trim(),
				configProps.getProperty(MetadataSourceConfigFieldsEnum.apiKey.toString()).trim()));

		if(LOG.isDebugEnabled()) {
			LOG.debug("Search response from TMS Program Image Search: [{}]", searchResponse);
		}

		// populate response with image information
		List<Map<String, Object>> imagesMap = new ArrayList<Map<String, Object>>();

		try {
			List<JSONObject> resultList = JsonPath.read(searchResponse, "$[*]");
			for(JSONObject resultMap : resultList) {
				Map<String, Object> imageInfo = new HashMap<String, Object>();

				String baseImagesPath = getImageBaseUrl(configProps);
				if(StringUtils.isNotEmpty(baseImagesPath)) {
					imageInfo.put(MetadataSourceConfigFieldsEnum.url.toString(),
baseImagesPath + resultMap.get("uri"));
				} else {
					imageInfo.put(MetadataSourceConfigFieldsEnum.url.toString(),
resultMap.get("uri"));
				}

				imageInfo.put(MetadataSourceConfigFieldsEnum.height.toString(),
resultMap.get("height"));
				imageInfo.put(MetadataSourceConfigFieldsEnum.width.toString(),
resultMap.get("width"));

				boolean languageMatch = true;

				// check languages
				// if(params != null && !params.isEmpty() && params.containsKey("locale")) {
				// // handle locale parameter
				// JSONObject resultLangObj = (JSONObject) resultMap.get("caption");
				// if(resultLangObj != null && resultLangObj.containsKey("lang")) {
				// String language = params.get("locale");
				// String resultLang = (String) resultLangObj.get("lang");

				if(languageMatch) {
					imagesMap.add(imageInfo);
				}
			}

		} catch(InvalidPathException ex) {
			LOG.error("Failed to read images from TMS API.", ex);
			throw new MetadataSourcePluginException(ex);
		}

		return imagesMap;
	}

	/**
```

```
	 * {@inheritDoc}
	 */
	@Override
	public MetadataDetails fetchDetails(String identifier, String locale) throws MetadataSourcePluginException,
IOException {
		LOG.debug("Begin TMS fetchDetails(): identifier=[{}], locale=[{}].", identifier, locale);

		Long time = System.currentTimeMillis();

		if(StringUtils.isEmpty(identifier)) {
			LOG.error(ID_CANT_BE_NULL);
			throw new MetadataSourcePluginException(ID_CANT_BE_NULL);
		}

		if(StringUtils.isBlank(locale)) {
			LOG.error(LOCALE_CANT_BE_NULL);
			throw new MetadataSourcePluginException(LOCALE_CANT_BE_NULL);
		}

		// get config and invoke TMS service for metadata details
		Properties configProps = getConfig().getBaseConfig();
		String searchResponse = fetchHttpBody(MessageFormat.format("{0}/programs/{1}?api_key={2}",
				configProps.getProperty(MetadataSourceConfigFieldsEnum.url.toString()).trim(),
identifier.trim(),
				configProps.getProperty(MetadataSourceConfigFieldsEnum.apiKey.toString()).trim()));

		LOG.debug("fetchDetails() response from TMS: [{}]", searchResponse);

		// populate response with metadata details
		MetadataDetails metadataDetails = new MetadataDetails();
		metadataDetails.setIdentifier(identifier);
		metadataDetails.setLocale(locale);

		// fetch image relative url and build absolute path
		try {
			String imageUrl = JsonPath.read(searchResponse, "$.preferredImage.uri");
			String baseImagesPath = getImageBaseUrl(configProps);
			if(StringUtils.isNotEmpty(baseImagesPath)) {
				metadataDetails.setImageUrl(baseImagesPath + imageUrl);
			} else {
				metadataDetails.setImageUrl(imageUrl);
			}
		} catch(InvalidPathException ex) {
			LOG.debug("Couldn't get preferredImage.uri from TMS", ex);
		}

		// collect plugin data in format: tmsKey=tmsRawValue
		Map<String, Object> values = metadataDetails.getRawValues();
		for(MappingConfigField field : MappingConfigUtil.getMappingConfigFields(getConfig().getFieldMapping()))
{
			String fieldName = field.getProviderFieldName();
			if(!values.keySet().contains(fieldName) && StringUtils.isNotEmpty(fieldName)) {
				try {
					values.put(fieldName, JsonPath.read(searchResponse, fieldName));
				} catch(InvalidPathException ex) {
					LOG.debug("Couldn't read from TMS a value for key: [{}]", fieldName);
				}
			}
		}
		String duration = convertDuration(values.get(DURATION_FIELD));
		if(duration != null) {
			values.put(DURATION_FIELD, duration);
		}

		LOG.debug("End fetchDetails() {}. MetadataDetails = [{}]", (System.currentTimeMillis() - time) +
TimeUtil.MSEC, metadataDetails);

		return metadataDetails;
	}

	/**
	 * {@inheritDoc}
```

```
         */
        @Override
        public Boolean providesMetadataSearch() {

                 LOG.debug("Begin providesMetadataSearch()");

                 Long time = System.currentTimeMillis();

                 LOG.debug("End providesMetadataSearch() {}" + (System.currentTimeMillis() - time) + TimeUtil.MSEC);

                 return Boolean.TRUE;
        }

        /**
         * {@inheritDoc}
         */
        @Override
        public Boolean providesImageSearch() {

                 LOG.debug("Begin providesImageSearch()");

                 Long time = System.currentTimeMillis();

                 LOG.debug("End providesImageSearch() {}", (System.currentTimeMillis() - time) + TimeUtil.MSEC);

                 return Boolean.TRUE;
        }

        /**
         * {@inheritDoc}
         */
        @Override
        public List<Integer> supportedPageSizes() {
                 return SUPPORTED_PAGE_SIZES;
        }

        /**
         * {@inheritDoc}
         */
        @Override
        public ProviderFieldFactory getProviderFieldFactory() {
                 return TMS_PROVIDER_FIELD_FACTORY;
        }

        @Override
        public List<MetadataConfigurationKey> getConfigurationsKeys() {
                 return CONFIG_FIELDS;
        }
}
```

Building the Plug-in

Follow the steps below to build a custom Metadata Search Plug-in:

1. Navigate to **Producer > Setup > Metadata Source Plugins**.
2. Click **Add New.**
3. Enter the following configuration information into the fields provided:
   - **a** Class: `vms.plugins.metadata.augmentation.tms.SampleMetadataSourcePlugin`
   - **b** Field Mappings: browse to and select "`tmsMapping.xml`"
   - **c** Locales: en_US
   - **d** Available fields: select all.
   - **e** Click **Confirm**.
   - **f** Under Configurations, select "imagesBaseUrl" from the Key drop-down menu.

g Enter "`http://csco.tmsimg.com/`" as the value.

h Select "apiKey" form the Key drop-down menu.

i Enter "9tvmn7xbfk6ghc2uwgktqz43" as the value.

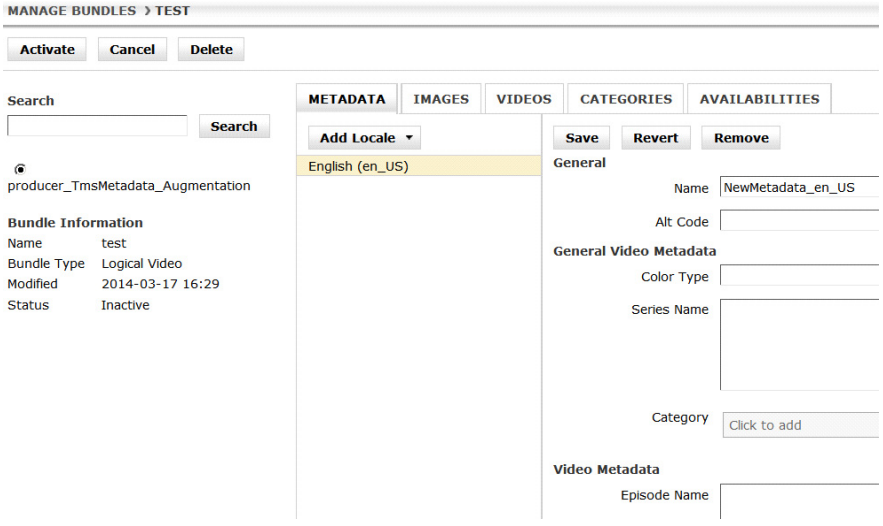
j Select "url" from the Key drop-down menu.

k Enter "`http://data.tmsapi.com/v1`" as the value.

4 Navigate to **Producer > Manage Bundles.**

5 Create a logical video bundle using the "`common_logical_video.xml`" template.

6 Open the new bundle.



Chapter 9

Customizing Search Manager Output

Overview

The following chapter describes how EPG search results are generated, and then explains how to customize that search output to meet the needs of your particular deployment.

Custom output formats allow you to override and customize default XML output from the Search Manager API. For example, the JSON (JavaScript Object Notation) format may be used instead of the XML default. API output can be presented in any available format, as long as the optional `formatCode` parameter is specified in the REST call. When the `formatCode` parameter is not specified, the output format will default to XML.

Search Process Overview

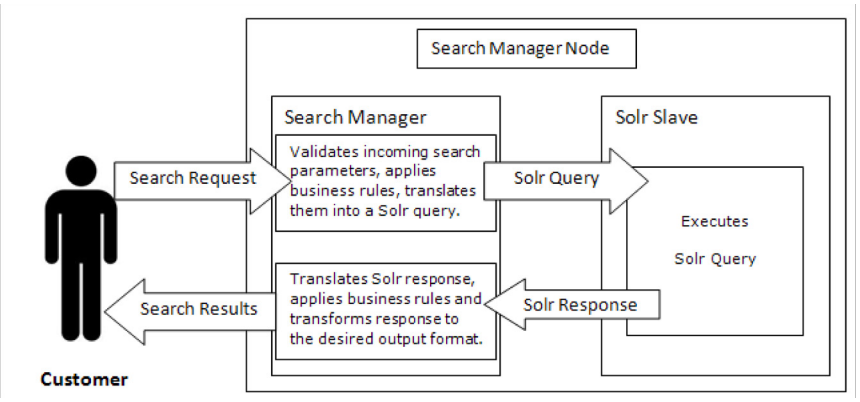
Two key nodes are used to generate the required information, and then to make it available for EPG searches:

- **Index Master Node** - generates an index of transformed EPG and VoD data and makes it available for use by Solr Slaves. This server is referred to as the Solr Master.
- **Search Manager Node** - enables searches to be performed. This node consists of a Search Manager component that queries a Solr Slave component via a REST API.

Search Manager Node does the following:

1. Receives search requests from end users.
2. Validates incoming search parameters.
3. Translates the search parameters into a Solr query by applying various business rules at the query transform and result level.
4. Executes the query on a Solr server.
5. Transforms Solr results (represented by a `SolrDocumentList` instance) into an output format by applying various custom business rules.

Figure 2 Search Process Overview



Custom Output Format Types

The procedure for creating a custom output depends on the format type.

There are two types of custom output formats used for transformations:

1 **Java-based formats** - are faster, lighter and more flexible. For more information go to “Creating Java-based Custom Output”.

Note The terms “Java-based Transformer” and “Java-based Output Format” will be used interchangeably within this document.

---

2 **XSLT-based formats** - are easier to implement. For further details refer to “Creating XSLT-based Custom Output”.

Note XSLT-based format types are more resource intensive than Java-based formats because they take input from default Java transformers. Within a single call both transformations are automatically invoked, starting with the default Java transformer and then continuing with the XSL transformation.

After the custom output format is deployed, you can retrieve the response in the desired format by specifying the `formatCode` parameter in the REST request. To find the appropriate default Java-based transformer related to each Public API call, refer to the “Transformer Usage” on page 135.

All default Java-based transformers produce output using the following XML template:

Figure 3 Default XML Template

```
<?xml version="1.0" encoding="UTF-8"?>
<xml>
    <requestElement totalFound="25" countReturned="3" countRequested="20"
  startRequested="0" />
    <documentElements>
        <documentElements>
            <field>1</field>
            <multivaluedFieldWrapper>
                <multivaluedField>value1</multivaluedField>
                <multivaluedField>value2</multivaluedField>
            </multivaluedFieldWrapper>
```

```
        </documentElements>
    <documentElements>
…
    </documentElements>
    </documentElements>
</xml>
```

Based on the "Default XML Template" on page 132, the `ContentDefaultJavaTransformer` produces the following output:

Figure 4 Default ContentDefaultJavaTransformer Output

```
<xml>
    <request totalFound="4" countReturned="4"countRequested="2000"startRequested="0"/
>
    <contents>
        <content>
            <et>EPG</et>
            <epn>2</epn>
            <ept>A Special Edition of 20/20: A Modern Fairytale</ept>
            <peid>EP000000211371</peid>
              <gs>
                <g>Newsmagazine</g>
                <g>Thriller</g>
            </gs>
                  <pid>abc15</pid>
                  <ts>Twenty by Twenty</ts>
                  <pt>Series</pt>
                  <srseid>eb306e56-ce56-5678-c94f-ed6ad6573923</srseid>
                  <seaid>SH005</seaid>
                  <epsn>2B</epsn>
                  <epasn>TV51</epasn>
                  <dss>A look back at the much-watched wedding.</dss>
                  <is>images</is>
        </content>
        <content>
                  <d>6180</d>
                  <et>EPG</et>
                  <peid>MV000000020000</peid>
                  <gs>
                      <g>Thriller</g>
                  </gs>
                  <pid>119074</pid>
                  <flags>fm</flags>
                  <rmp>R</rmp>
                  <pt>Feature Film</pt>
                 <dss>Janitor (William Hurt) seduces TV newswoman (Sigourney Weaver)
                   with a lie that backfires.</dss>
                  <ts>Eyewitnes1</ts>
                  <sn>4</sn>
                  <srseid>ba092b34-dc95-2619-a47f-af9fb3126378</srseid>
        </content>
    </contents>
</xml>
```

Understanding Solr Technology

Solr, built on top of Lucene, is an open source enterprise search platform that enables functionality to create sophisticated, high performance search applications. In a Media Suite context, Solr is responsible for serving search results for Search Manager requests via a REST API.

At the most basic level, Solr is technology that you provide with a large amount of information. The power of Solr is that it can generate highly customizable indexes so that it can quickly and precisely retrieve the exact results you need in the future.

The following Web sites provide detailed information on Solr technology:
http://lucidworks.lucidimagination.com/display/solr/Apache+Solr+Reference+Guide

and

http://wiki.apache.org/solr/

Solr Concepts

The following section explains important concepts related to Solr.

Solr Documents

When creating Java-based transformations, you must work with, and have an understanding of Solr documents. Solr sees everything as a *document*, or a set of data that describes something. These documents are composed of name value pairs that comprise *fields*, which contain specific information about the item being described. Each field must be correctly defined at the outset by specifying a field type so that Solr knows how to interpret the information when it is building an index, or later, querying the data. When Solr returns search results, it returns them in the form of a Solr document, which must then be transformed into a custom format for use.

Solr Schemas

Solr Schemas define the structure of Solr documents. The schema file contains all the details about the fields that your document can use and how those fields should respond when queried.

Schema files includes information about:

- field names
- field types (such as simple, multivalued, or dynamic)
- field data types (such as string or integer)
- indexed fields (users can search the content of these fields)
- stored fields (fields whose values are returned in Solr responses)

Details about the Solr schema file can be found at http://wiki.apache.org/solr/SchemaXml.

Solr Cores

A Solr Core consists of one running instance of a Solr Index along with its related configuration files. These Solr Cores define specific combinations of fields that are searchable within a Solr document. The same Solr Schema can be reused by many different Solr cores.

Note Search Manager nodes only work with known cores as depicted in the following table. Within these cores, however, you can add additional fields as necessary. For more information see "Customizing Searchable VoD Content".

The following table summarizes the default Media Suite Solr Cores:

Table 7 Media Suite Solr Cores

Solr Core	Solr Schema Path	Description
regions	${install.solr.dir}/slave_home/conf/regions-schema.xml	Contains detailed information about regions.
stations	${install.solr.dir}/slave_home/conf/stations-schema.xml	Contains detailed information about stations.
schedules-$[languageCode]_$[countryCode]	${install.solr.dir}/slave_home/conf/schedules-schema.xml	Contains detailed information about schedules.
corrections	${install.solr.dir}/slave_home/conf/schedules-schema.xml	Is an alias to the schedules core. This alias was added to have the ability to distinguish a schedule from a schedule correction document.
programs-${languageCode]_$[countryCode}	${install.solr.dir}/slave_home/conf/programs-schema.xml	Contains detailed information about programs.
vod-${languageCode]_$[countryCode}	${install.solr.dir}/slave_home/conf/vod/vod-schema-${languageCode}.xml	Contains detailed information about Video On Demand.

Details about all registered Solr Cores, in a particular Solr instance can be found in the `${install.solr.dir}/slave_home/solr.xml` file.

Transformer Usage

Default Java Transformations are provided for each API call. The following table summarizes the Solr Cores and related Transformers that are used by Public REST API calls:

Table 8 Transformer Usage Table

Public API call	Solr Cores	Default Java Transformer
/opencase/sm/resource/rest/regions	regions	DefaultJavaTransformer
/opencase/sm/resource/rest/channel	stations	ChannelDefaultJavaTransformer
/opencase/sm/resource/rest/channels	stations	ChannelsDefaultJavaTransformer
/opencase/sm/resource/rest/schedules	schedules	SchedulesDefaultJavaTransformer
/opencase/sm/resource/rest/schedule	schedules, stations, programs	ScheduleDetailsDefaultJavaTransformer

Table 8 Transformer Usage Table

Public API call	Solr Cores	Default Java Transformer
/opencase/sm/resource/rest/modifiedschedules	schedules, corrections	ModifiedSchedulesJavaTransformer
/opencase/sm/resource/rest/programschedule	schedules	ProgramScheduleDefaultJavaTransformer
/opencase/sm/resource/rest/contentdetails	programs, VoD	DefaultJavaTransformer
/opencase/sm/resource/rest/typeahead	programs, VoD	ContentDefaultJavaTransformer
/opencase/sm/resource/rest/content	programs, VoD	ContentDefaultJavaTransformer
/opencase/sm/resource/rest/related	programs, VoD	ContentDefaultJavaTransformer

Configuring Returned Fields

As previously mentioned, Search Manager calls the Solr Slave, which then returns a set of Solr documents that match the search criteria within the request. Solr documents contains name-value pairs consisting of a key with a related value, and each one of these pairs comprises a Solr field.

Search Manager can only retrieve a subset of stored document fields from Solr. The default Java transformers will then send all fields that were fetched into the output. Each API call has its own default set of fields that are output. Those fields will be described in "Search Manager API Details".

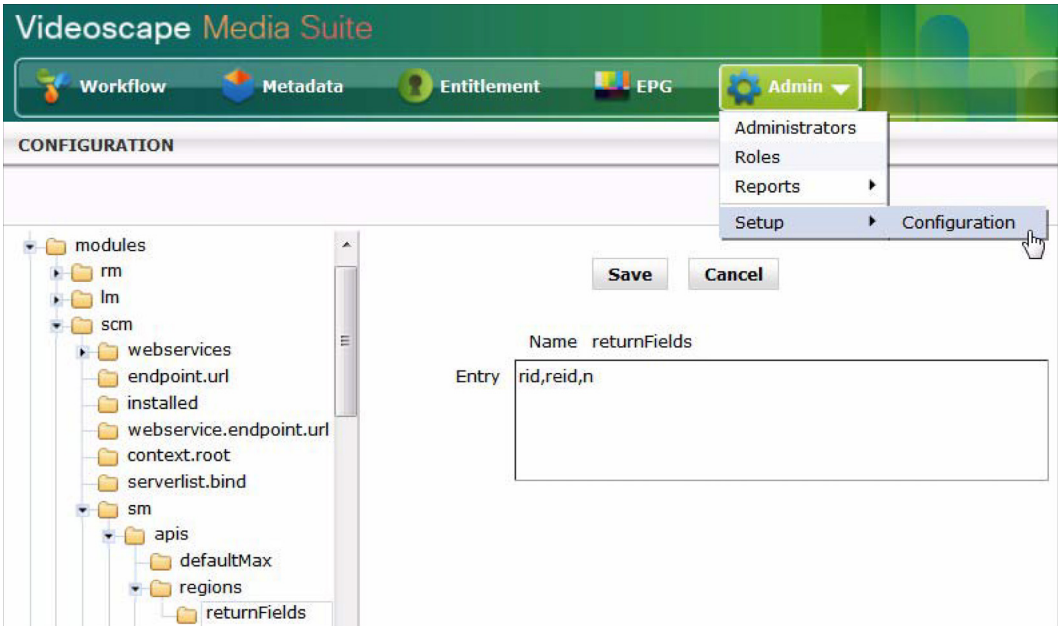
Filtering out fields at the custom output format level is not advised. Instead, developers should utilize the Media Suite user interface to configure the desired return fields for each API call.

As an example, we will look at the `/opencase/sm/resource/rest/regions` API call. According to the API, this call returns the following fields by default: `rid`, `reid`, `n`, `ds`, `loc`, `tz`, `sys`, `rsc`, `cmdt`, and `mdt`. Assuming that you only need the `rid`, `reid`, and `n` fields in the response, you would configure the system in the following manner.

To configure returned Search Manager fields:

1. In the Media Suite interface, navigate to **Admin > Setup > Configuration**.
2. In the configuration tree structure, open **modules > scm > sm > apis > regions > returnFields**.
3. Type the required fields (in this case `rid,reid,n`) into the **Entry** textbox.

Figure 5 Configuring Search Manager Return Fields



4 Click **Save**.

5 Wait for one minute so that the changes can propagate throughout the system, then flush the Search Manager cache via the Admin interface. Search Manager will now return only the specified fields in its responses.

Note Fields that are marked as "not-stored" are not returned from the Solr server, but are used strictly for searching and sorting purposes. To verify the fields that are stored, consult the schema files in the table "Media Suite Solr Cores".

XML Values in Solr

The majority of fields in Solr are of type String, but some fields contains XML values only. The following table lists those fields.

Table 9 XML Values in Solr

Solr Core	Field Name	Field Description	Field Value Example	Description
STATIONS	bxml	Station Bundle XML	See `\Custom Output\bundlexml.xml` in the sample code	These fields are generated by external modules and stored in EPG module. Their structure is variable and can be customized. For more details, see "Customizing Stations for EPG Channel Lineups".
	cbxml	Channel Bundle XML		
	pxml	Station Product XML	See `\Custom Output\productxml.xml` in the sample code	
	cpxml	Channel Product XML		

Table 9 XML Values in Solr

Solr Core	Field Name	Field Description	Field Value Example	Description
STATIONS, SCHEDULES	sls	Station images	http://www.host.com/1http://www.host.com/2	• attribute 'w' stands for image width • attribute 'h' stands for image height • attribute 'p' stands for image primary flag • attribute 'd' stands for image default flag
STATIONS	cfs	Station Custom fields	<c><n>name1</n><v>value1</v></c><c><n>name2</n><v>value2</v></c>	• element 'n' stands for custom field name • element 'v' stands for custom field value
PROGRAMS		Program Custom fields		
SCHEDULES		Schedule Custom fields		
PROGRAMS	cntrs	Program contributors	<c o="1" r="Host"><f>Bob</f><l>Littell</l></c>	• attribute 'o' stands for contributor rank • attribute 'r' stands for contributor role • element 'f' stands for contributor first name • element 'l' stands for contributor last name
PROGRAMS	paids	Program additional identifiers	<paid n="tmsid" v="EP006609610159" /><paid n="tmscid" v="SH006609610000" />	• attribute 'n' stands for identifier name • attribute 'v' stands for identifier value
PROGRAMS, VOD, SCHEDULES	is	Program images	uri1uri2	• attribute 'w' stands for image width • attribute 'h' stands for image height • attribute 'c' stands for image category

To parse any of the returned fields you can use:

1 The XML processing (JAXP) classes within the JDK. For details about JAXP, consult http://jaxp.java.net/

2 Third-party libraries, as long as they are deployed with the custom formats as part of the JAR file or deployed into the `${JBOSS_HOME}/server/${serverName}/deploy/SearchManager.war/WEB-INF/lib` directory on each Search Manager node.

Adding Custom Fields

It is possible to add new fields into Search Manager. Certain customizations require metadata that is not part of the default model be added to the indexes. This metadata must be searched for and retrieved by the end user. There are two available approaches for adding this data to the indexes.

The first approach is to use a set of predefined dynamic fields and stored fields. These fields are designated with a prefix `cf_[type]_*` and `cf_*` and are used to create custom fields when adding a solr document. For example, to add new metadata to the VoD core as a numeric value that will be available in search results, a new field can be added to the solr document being submitted for indexing. This field would look like this:

```
<fieldname="cf_num_mynumber">123</field>
```

It will be added to the index as "cf_num_mynumber" and as a result can be referenced in Search Manager queries. In addition, configuration can be altered to return this field in responses from Search Manager.

Extensible custom fields are only available in the VoD, Programs and Schedules schemas. Regions and Stations do not have custom fields built in and require another approach, outlined below.

To add the custom field for Regions and Stations:

1. You need to ascertain the Solr schema that should be modified.
   According to the “Transformer Usage” on page 135, we can see that the `/opencase/sm/resource/rest/regions` API call uses the Solr `regions` core.
   Next, according to the "Media Suite Solr Cores" table on page 135, we can ascertain that the `${install.solr.dir}/slave_home/conf/regions-schema.xml` file should be modified.
2. Add the field definition into the Solr schema. The field definition should be:
   a. “stored,” so that we can see it in the output
   b. indexed, so that we can search on it
   c. added into the “fields” section of the schema file as follows:

```
…
<fields>
    …
    <field name="customField" stored="true" indexed="true" type="string"/>
</fields>
```

3. Populate the Solr field with data.
4. To see the field in the Search Manager output, add it to the return fields that were configured for the `/opencase/sm/resource/rest/regions` API call. For details on this process, see “Configuring Returned Fields” on page 136.

Note Scoring is the highly sophisticated logic behind Solr searches that ranks search items by their number of positive hits. This logic uses a combination of boolean, vector space, and fuzzy math to generate its results.

Norms are applied to allow for index time boosts and field length normalization. This allows you to add scoring boosts to fields at index time, which makes shorter documents score higher than longer ones. Norms are memory intensive and should be turned off for dynamic text fields to significantly reduce the index size.

Caching API Responses

To improve performance by reducing the query response time, all Public API responses (except for `/opencase/sm/resource/rest/version`) are cached. Once an API call has succeeded, its response is cached and all subsequent requests, which produce the same response, return data from the cache.

Note `/opencase/sm/resource/rest/version` is cached in EPG 4.0.2 and later releases.

Caching is configured via Java Servlet Filters in:
`${JBOSS_HOME}/server/${serverName}/deploy/SearchManager.war/WEB-INF/web.xml`
A typical cache configuration would be as follows:

```
<filter>
    <display-name>Cache for getRegions API call</display-name>
    <filter-name>getRegionsCacheFilter</filter-name>
    <filter-class>com.extend.cache.filter.CachingFilter</filter-class>
    <init-param>
      <param-name>cacheName</param-name>
      <param-value>REGIONS</param-value>
    </init-param>
    <init-param>
      <param-name>keyGenerator</param-name>
      <param-value>com.extend.cache.key.KeyGenerator</param-value>
    </init-param>
    <init-param>
      <param-name>useParams</param-name>
      <param-value>start,count,regionId,externalId,servedId,formatCode</param-value>
    </init-param>
</filter>

<filter-mapping>
    <filter-name>getRegionsCacheFilter</filter-name>
    <url-pattern>/resource/rest/regions</url-pattern>
</filter-mapping>
```

Where:

- **filter/filter-class** – specifies the fully qualified class name of the filter. This value must not be changed.
- **filter/init-param/cacheName** – specifies the name of cache where cached responses reside. This value should not be changed.
- **filter/init-param/keyGenerator** – specifies the fully qualified class name of the cache key generator. The aim of the key generator is to produce a key under which a response is stored in the cache. The generated key value should be the same for all requests that return the same response. The value of this parameter can be changed to the custom class name, but the key generator must implement the `com.extend.cache.key.ICacheKeyGenerator` interface.

The default `com.extend.cache.key.KeyGenerator` generator can create a key based on a combination of any of the following parts of the HTTP request:

- protocol
- host
- port
- URI
- any parameter found in the query string

The parts of the HTTP request used for key construction are specified by providing the following configuration values:

- **useProtocol** – specifies whether the cache key depends on the request protocol value. The default is false.
- **useHost** – specifies whether the cache key depends on request host value. The default is false
- **usePort** – specifies whether the cache key depends on request port value. The default is false
- **useParams** – a comma delimited list of parameter names and their values to include in the key. If not set, only the URI will be used for key generation.
- **ignoreParams** – a comma delimited list of parameter names that should be excluded from key generation. This value is only examined if an "<all>" value is set for `useParams`.

- **filter/init-param/useParams** – specifies a comma delimited list of parameter names and their values to be included in key. By default, all Public API call parameters are listed.

Warning Be very careful when changing the parameter list as improperly removing a single parameter from the list may result in inconsistent application behavior.

- **filter-mapping/filter-name** – specifies the name of a filter and should match filter/filter-name.
- **filter-mapping/url-pattern** – specifies the URL pattern of the API call to cache responses for.

Creating Custom Output

The default structure and format of Search Manager output often needs to be modified in order to match client needs, such as using different field names or including new fields. This functionality is performed via custom output formats, which can be created by either implementing custom Java code or XSL transformations. In turn, the process for creating a custom output format will vary depending on whether the format is a Java or XSLT-based implementation.

In general, for Java-based custom output formats, you need to do the following:

1. Implement the `com.cisco.videoscape.sm.transformation.Transformer` interface.
2. Define an empty constructor in the custom class.
3. Compile the custom class.
4. Configure the custom class on the Discovery > Transformer page.Copy the custom class to the server class-path.

5 Activate.

In general, for XSLT-based custom output formats:

1 Create the XLST that will provide the required transformations. A sample file is provided under `custom-sm-plugin\src\main\resources\SampleXSLTTransformer.xsl`. The file is named SampleXSLTTransformer.xsl.

2 Configure the XSLT on the Discovery > Transformer screen.

3 Activate.

Note XSLT-based formats are heavier than Java-based formats because they take input from default Java transformers. As a result, you are invoking two transformations with a single call: the default Java transformer, and the XSL transformation.

Extending Output Formats

The diagram that follows depicts Java classes that can be extended to customize Search Manager output for Java-based transformers. The depicted classes will be described in more detail after the diagram.

Figure 6 Classes for Extending Search Manager Output

Output Format Classes

The following section provides details on the classes depicted in "Classes for Extending Search Manager Output" (Figure 6).

Table 10 Solr Field Types

Class Name
com.extend.opencase.occ.client.search.solr.SolrFieldType
Description
Represents a Solr field type (as found in the Solr schema). Possible values include:
• SolrFieldType.SIMPLE – represents Solr simple fields. A single field can contain only a single value for each Solr document. Example from Solr schema: `<field name="rid" stored="true" indexed="true" type="string" required="true" />`. Using the above declaration you can store the following field in Solr document: rid=″1″, etc. • SolrFieldType.MULTIVALUED – represents Solr multivalued field. That is, a single field can contain multiple values per Solr document. Example from Solr schema: `<field name="g" stored="true" indexed="true" type="text_wildcard"` **multiValued**`="true" />`. Using the above declaration allows you to store the following fields in Solr document: g=″comedy″, g=″drama″, etc.
• SolrFieldType.DYNAMIC - represents Solr dynamic fields. Using the dynamic field type allows you to create field rules that Solr will follow to determine the data type that should be used. For instance, when Solr is given a field name that is not explicitly defined, however, the field name type matches a prefix or suffix used in a dynamicField declaration. Example from Solr schema: `<dynamicField name="cl_*" stored="true" indexed="true" type="text_wildcard" omitNorms="true"/>`. Using the above declaration allows you to store the following fields in Solr document: cl_1=″label1″, cl_2=″label2″, cl_3=″label3″, etc.
• SolrFieldType.DYNAMIC_MULTIVALUED - represents Solr dynamic multivalued fields. Fields of the Dynamic_MultiValue type combine features of SolrFieldType.DYNAMIC and SolrFieldType.MULTIVALUED types. Example from Solr schema: `<dynamicField name="cn_*" stored="true" indexed="true" type="text_wildcard" omitNorms="true"` **multiValued**`="true"/>`. Using the above declaration allows you to store the following fields in Solr document: cn_1=″11″, cn_1=″12″, cn_2=″2″, etc.

Table 11 Solr Field Data Types

Class Name
com.extend.opencase.occ.client.search.solr.SolrFieldDataType
Description
Represents Solr data types that are found in the Solr schema. Possible values include:
• SolrFieldDataType.INT – for integer data type Solr fields.
• SolrFieldDataType.DATE – for date type Solr fields.
• SolrFieldDataType.BOOLEAN – for boolean type Solr fields.
• SolrFieldDataType.STRING – for string type Solr fields with special characters (such as < or & or > or ,) that would require escaping.
• SolrFieldDataType.RAW_STRING – for string type Solr fields that are either already escaped or contain data that does not require escaping.
• SolrFieldDataType.DOUBLE – for double type Solr fields.

Table 12 Solr Cores

Class Name
com.extend.opencase.occ.client.search.solr.SolrCore
Description
Represents an instance of a Solr core. For details refer to the Solr Cores table "Media Suite Solr Cores" on page 135. This class contains useful methods that receive information about: • all fields. • all dynamic fields. • whether a field exists with a specific name

Table 13 Solr Fields

Interface Name
com.extend.opencase.occ.client.search.solr.schema.SolrField
Description
Represents Solr fields as found in the Solr schema. This interface contains the following information:
• Field name Specifies Solr field names as they appear in the Solr schema. For example, rig, n, ds.
• Solr field type Specifies Solr field types as they appear in the Solr schema. See "Solr Field Types" on page 144.
• Solr field data type Specifies Sold field data types as they appear in the Solr schema. See "Solr Field Data Types" on page 145.

Table 13 Solr Fields

Interface Name
• Solr sorting field name This optional value specifies the Solr field name that is used for sorting. It was introduced as a workaround to address a limitation inherent in Solr sorting. For example, Solr cannot sort over a tokenized field. In such cases a new non-tokenized field is introduced in the Solr schema and is used as the sort field name.

Table 14 Solr Core Programs Schema Fields

Class Name
com.extend.opencase.occ.client.search.solr.schema.ProgramSchemaField
Description
Represents all `programs` Solr core fields. Every field in the current enumeration belongs to the `programs` Solr core.

Table 15 Solr Core Regions Schema Fields

Class Name
com.extend.opencase.occ.client.search.solr.schema.RegionSchemaField
Description
Represents all `regions` Solr core fields. Every field in the current enumeration belongs to the `regions` Solr core.

Table 16 Solr Core Schedules Schema Fields

Class Name
com.extend.opencase.occ.client.search.solr.schema.ScheduleSchemaField
Description
Represents all `schedules` Solr core fields. Every field in the current enumeration belongs to the `schedules` Solr core.

Table 17 Solr Core Stations Schema Fields

Class Name
com.extend.opencase.occ.client.search.solr.schema.StationSchemaField
Description
Represents all `stations` Solr core fields. Every field in the current enumeration belongs to the `stations` Solr core.

Table 18 Solr Core VoD Schema Fields

Class Name
com.extend.opencase.occ.client.search.solr.schema.VodSchemaField
Description
Represents all `VoD` Solr core fields. Every field in the current enumeration belongs to the `VoD` Solr core.

Table 19 Transformer

Class Name	
com.cisco.videoscape.sm.transformation.Transformer	
Description	
Responsible for transforming entities to an outgoing format.	
Method Signature	
void transform(OutputStream outputStream, Map<SolrCore, SolrDocumentList> entities, Map<String, String> parameters) throws TransformerException;	
Description	Transforms entities to an outgoing format and writes the results to the provided OutputStream.
Parameter Name	**Description**
outputStream	The stream that is used to print the transformed format to.
Type:	java.io.OutputStream
Nullable:	False
entities	`SolrDocumentList` per SolrCore to transform. For details, refer to "Search Process Overview" on page 131.
Type:	java.util.Map<SolrCore, SolrDocumentList>
Nullable:	False
parameters	Metadata such as region ID, number of found documents, start offset, count of requested documents, initial search request information, etc. Part of each entry comes from the Search Manager request, while another part comes from Solr response.
Type:	java.util.Map<String, String>
Nullable:	True
Exceptions	com.cisco.videoscape.sm.transformation.TransformerException For use if any exception occurs during the transformation process.

Table 20 getMediaType Method

Method Name
String getMediaType();
Description
Gets the media type for the transformer implementation.

Table 21 AbstractBase Transformer Class

Class Name
com.cisco.videoscape.sm.transformation.AbstractBaseTransformer implements Transformer
Description
Contains methods to transform fetched entities to an outgoing format.

Note For each of the following classes, to see which API calls use the current transformer by default, refer to "Transformer Usage Table" on page 135. For sample output, refer to "Search Manager API Details".

Table 22 ModifiedSchedulesJavaTransformer Class

Class Name
com.cisco.videoscape.sm.transformation.ModifiedSchedulesJavaTransformer extends AbstractBaseTransformer
Description
Transforms entities to the modified default schedule output format.

Table 23 ProgramScheduleDefaultJavaTransformer Class

Class Name
com.cisco.videoscape.sm.transformation.ProgramScheduleDefaultJavaTransformer extends AbstractBaseTransformer
Description
Transforms entities to the default program schedule output format.

Table 24 DefaultJavaTransformer Class

Class Name
com.cisco.videoscape.sm.transformation.DefaultJavaTransformer extends AbstractBaseTransformer
Description
Default Java Transformer that transforms entities to the outgoing format using pure Java code. Transformations follow these rules: • No attributes are printed to the documents section in the outgoing feed. • Field name becomes an XML element name for SolrFieldType.SIMPLE or SolrFieldType.DYNAMIC field types. • Field name becomes an XML element name for SolrFieldType.MULTIVALUED or SolrFieldType.DYNAMIC_MULTIVALUED field types. One XML element per field value. In the output, all multivalued fields are wrapped in an XML element with name = multivalued field name + 's' suffix.

Table 25 Channels Default JavaTransformer class

Class Name
com.cisco.videoscape.sm.transformation.ChannelsDefaultJavaTransformer extends DefaultJavaTransformer
Description
Transforms entities to the default channel output format.

Table 26 Content Default JavaTransformer class

Class Name
com.cisco.videoscape.sm.transformation.ContentDefaultJavaTransformer extends DefaultJavaTransformer
Description
Transforms entities to the default content output format.

Table 27 Schedules Default JavaTransformer class

Class Name
com.cisco.videoscape.sm.transformation.SchedulesDefaultJavaTransformer extends DefaultJavaTransformer
Description
Transforms entities to the default schedule output format.

Table 28 Schedule Details DefaultJavaTransformer class

Class Name
com.cisco.videoscape.sm.transformation.ScheduleDetailsDefaultJavaTransformer extends DefaultJavaTransformer
Description
Transforms entities to the default program schedule details output format.

Custom Output Sample

In the following example, we are going to assume that customized output in Figure 7 needs to be generated from the Public API `/opencase/sm/resource/rest/regions`. (For details on this REST method, see "Get Regions".)

Figure 7 Sample Custom Output

```
<?xml version="1.0" encoding="UTF-8"?>
<xml>
  <searchRequestInfo>
    <totalFound>1868</totalFound>
    <countReturned>1</countReturned>
    <countRequested>1</countRequested>
    <startRequested>0</startRequested>
    <requestInfo>/opencase/sm/resource/rest/regions?start=2&count=1
    </requestInfo>
  </searchRequestInfo>
  <regions>
    <region id="50000" externalId="AL01402-X">
      <deviceType>X</deviceType>
      <description>Florence Region Description</description>
      <name>Florence</name>
      <location>Anchorage</location>
      <modifiedDate>2012-01-03T14:03:51.025Z</modifiedDate>
      <timezone>Central Observing</timezone>
      <systemType>System Type</systemType>
      <areas>
        <area>35617</area>
        <area>35630</area>
        <area>35631</area>
      </areas>
    </region>
    <region id="50001" externalId="AL01403-X">
      <name>Florence2</name>
    </region>
  </regions>
</xml>
```

In the customized output format, field names are more expressive than those in the default output. Also, the custom format contains the `requestInfo` element that represents information about the initial search request. Each API call places this information into a parameters map and passes it into the specified output format (or the corresponding `DefaultJavaTransformer` if no output format is specified). As a result, you have access to the initial search request parameters in your custom

output format. This allows you to, for example, produce different output based on custom parameters within your request. For implementation details, refer to the provided custom output format examples.

The following table shows how default elements are mapped to new elements for this example:

Table 29 Mapping of Default Elements

Default Field Name	Transformed Field Name
request	searchRequestInfo
r	region
rid	id
reid	externalId
cmdt	deviceType
ds	description
n	name
loc	location
mdt	modifiedDate
tz	timezone
sys	systemType
rscs	areas
rsc	area

The sections that follow will explain in detail how to develop XSLT and then Java-based transformers that produce the sample output shown in “Sample Custom Output” on page 150.

Deploying Custom Output Code XSLT

After your new output format has been created, you are ready to upload and configure the new plugin to the system.

To deploy a custom output format:

1. Within the Media Suite User interface, navigate to **Discovery > Transformers > XSLT Transformers.**
2. Click **Add New.**
3. Enter a name, format code, and description for the transformer.

Note The Format Code for each transformer must be unique.

4. Click **Create and Edit.**
5. Specify a Media Type.
6. Upload the XSLT file.
7. Click **Save and Activate**.

Creating Java-based Custom Output

Java-based custom output should implement the `com.cisco.videoscape.sm.transformation.Transformer` interface. For the current implementation, we have decided to extend the `com.cisco.videoscape.sm.transformation.AbstractBaseTransformer` class to reuse its useful methods

The current sample of a custom Java-based Transformer does the following:

1. Extend `com.cisco.videoscape.sm.transformation.AbstractBaseTransformer` class to reuse its methods.
2. Define an empty constructor.
3. Transform the Solr field name into a new name according to the "Mapping of Default Elements" on page 151.
4. Write all parameters from the original URL request as child XML elements for the `requestInfo` element in output. Original parameters are made available to the XSLT from the child nodes of the `rid` Solr field value.
5. Every Solr document must be represented by an XML `region` element in the output. The `rid` and `reid` Solr field values must have `correspondingID` and `externalID` attributes for the region element.

   To view an example of a Java Transformer example, see the `RegionTransformerForPublicAPI.java` that is provided with the sample code.

Figure 8 Sample Java-Based Transformer

```
/**
 * Current Java Transformer is intended to be used for /sm/resource/rest/regions API call.
 * <p/>
 * Sample output:
 * <p/>
 * <pre>
 * {@code
 * <?xml version="1.0" encoding="UTF-8"?>
 * <xml>
 *       <searchRequestInfo>
 *                 <totalFound>1868</totalFound>
 *                 <countReturned>1</countReturned>
 *                 <countRequested>1</countRequested>
 *                 <startRequested>0</startRequested>
 *                 <requestInfo>/opencase/sm/resource/rest/regions?start=2&count=1</requestInfo>
 *       </searchRequestInfo>
 *       <regions>
 *                 <region id="50000" externalId="AL01402-X">
 *                           <deviceType>X</deviceType>
 *                           <description>Florence Region Description</description>
 *                           <name>Florence</name>
 *                           <location>Anchorage</location>
 *                           <modifiedDate>2012-01-03T14:03:51.025Z</modifiedDate>
 *                           <timezone>Central Observing</timezone>
 *                           <systemType>System Type</systemType>
 *                           <areas>
 *                                     <area>35617</area>
 *                                     <area>35630</area>
 *                                     <area>35631</area>
 *                                     <area>35632</area>
 *                                     <area>35633</area>
 *                                     <area>35634</area>
 *                                     <area>35661</area>
 *                                     <area>35662</area>
 *                                     <area>35653</area>
 *                                     <area>35654</area>
 *                                     <area>35660</area>
```

```
 *                                  <area>35674</area>
 *                          </areas>
 *                 </region>
 *                 <region id="12255" externalId="AL01403-X">
 *                          <name>Florence2</name>
 *                 </region>
 *        </regions>
 * </xml>
 * }
 * </pre>
 */
public class RegionTransformerForPublicAPI extends AbstractBaseTransformer {

        private static final Logger LOG = LoggerFactory.getLogger(RegionTransformerForPublicAPI.class);

        private static final String XML_DECLARATION = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>";
        private static final String RESPONSE_ROOT_ELEMENT_NAME = "xml";
        private static final String REQUEST_ELEMENT_NAME = "searchRequestInfo";
        private static final String REGIONS_ELEMENT_NAME = "regions";
        private static final String REGION_ELEMENT_NAME = "region";
        private static final String REGION_ID_FIELD_NAME = "rid";
        private static final String REGION_EXTERNAL_ID_FIELD_NAME = "reid";

        /**
         * Keeps information about how input field names should appear in output. Entries from this map will appear as
        attributes in element that describes Solr document in output (REGION_ELEMENT_NAME).
         */
        private static final Map<String, String> IN_OUT_FIELD_NAME_AS_ATTRIBUTES_MAPPING = new HashMap<String, String>();
        /**
         * Keeps information about how input field names should appear in output.
         */
        private static final Map<String, String> IN_OUT_FIELD_NAME_MAPPING = new HashMap<String, String>();

        static {
                IN_OUT_FIELD_NAME_AS_ATTRIBUTES_MAPPING.put(REGION_ID_FIELD_NAME, "id");
                IN_OUT_FIELD_NAME_AS_ATTRIBUTES_MAPPING.put(REGION_EXTERNAL_ID_FIELD_NAME, "externalId");

                IN_OUT_FIELD_NAME_MAPPING.put("n", "name");
                IN_OUT_FIELD_NAME_MAPPING.put("ds", "description");
                IN_OUT_FIELD_NAME_MAPPING.put("rsc", "area");
                IN_OUT_FIELD_NAME_MAPPING.put("mdt", "modifiedDate");
                IN_OUT_FIELD_NAME_MAPPING.put("cmdt", "deviceType");
                IN_OUT_FIELD_NAME_MAPPING.put("loc", "location");
                IN_OUT_FIELD_NAME_MAPPING.put("tz", "timezone");
                IN_OUT_FIELD_NAME_MAPPING.put("sys", "systemType");
        }

        /**
         * Default constructor is mandatory for custom Java based transformer.
         */
        public RegionTransformerForPublicAPI() {
                super();
        }

        @Override
        public String getMediaType() {
                return "text/xml";
        }

        @Override
        public void transform(final OutputStream outputStream, final Map<SolrCore, SolrDocumentList> entities, final
Map<String, String> parameters)
                        throws TransformerException {
                try {
                        // transform output stream to Writer to be able to use handy methods from
                        AbstractBaseTransformer
                        final Writer out = new BufferedWriter(new OutputStreamWriter(outputStream));
                        out.write(XML_DECLARATION);
                        writeXMLStartTag(out, RESPONSE_ROOT_ELEMENT_NAME, null);
                        // invoke overridden method to print all parameters as child XML elements not as attributes
                        writeRequestXMLElement(out, REQUEST_ELEMENT_NAME, parameters);
```

CUSTOM OUTPUT SAMPLE

```
                        final SolrDocumentList documents = entities.get(SolrCore.REGIONS);
                        if(documents != null) {
                                writeXMLStartTag(out, REGIONS_ELEMENT_NAME, null);
                                // iterate over all documents from Solr server.
                                for(final SolrDocument document : documents) {
                                        // prepare map of attributes for REGION_ELEMENT_NAME
                                        final Map<String, String> regionElementAttributes = new HashMap<String,
                                        String>(2, 1);
                                        final String idAttrName =
                                        IN_OUT_FIELD_NAME_AS_ATTRIBUTES_MAPPING.get(REGION_ID_FIELD_NAME);
                                        regionElementAttributes.put(idAttrName,
                                        document.getFieldValue(REGION_ID_FIELD_NAME).toString());
                                        final String externalIdAttrName =
                                        IN_OUT_FIELD_NAME_AS_ATTRIBUTES_MAPPING.get(REGION_EXTERNAL_ID_FIELD_NAME)
                                        ;
                                        regionElementAttributes.put(externalIdAttrName,
                                        document.getFieldValue(REGION_EXTERNAL_ID_FIELD_NAME).toString());

                                        writeXMLStartTag(out, REGION_ELEMENT_NAME, regionElementAttributes);
                                        // invoke overridden method to print only fields from
                                        IN_OUT_FIELD_NAME_MAPPING map
                                        writeDocument(out, document, SolrCore.REGIONS);
                                        writeXMLEndTag(out, REGION_ELEMENT_NAME);
                                }
                                writeXMLEndTag(out, REGIONS_ELEMENT_NAME);
                        }
                        writeXMLEndTag(out, RESPONSE_ROOT_ELEMENT_NAME);
                        out.flush();
                } catch(final IOException e) {
                        throw new TransformerException("Java Transformation exception occurred.", e);
                }

        }

        /**
         * Overridden version of method that writes to output all parameters as child XML elements, not as XML element
attributes.
         *
         * @param out writer to write XML element to
         * @param requestElementName name of XML element to write
         * @param parameters all parameters to print to output as child XML elements
         * @throws TransformerException in case of any exception
         */
        @Override
        protected void writeRequestXMLElement(final Writer out, final String requestElementName, final Map<String,
String> parameters)
                        throws TransformerException {
                writeXMLStartTag(out, requestElementName, null);
                for(final Map.Entry<String, String> entry : parameters.entrySet()) {
                        writeXMLElement(out, entry.getKey(), null, entry.getValue(), true);
                }
                writeXMLEndTag(out, requestElementName);
        }

        /**
         * Overridden version of method that writes to output only fields from IN_OUT_FIELD_NAME_MAPPING map. All fields
which are not present in
         * IN_OUT_FIELD_NAME_MAPPING map are ignored.
         *
         * @param out writer to write XML to
         * @param document <code>SolrDocument</code> to write
         * @param core specifies core <code>SolrDocument</code> belongs to
         * @throws TransformerException in case of any exception
         */
        @Override
        protected void writeDocument(final Writer out, final SolrDocument document, final SolrCore core) throws
TransformerException {
                for(final String fieldName : document.getFieldNames()) {
                        // iterate over document fields and print fields which are present in
                        IN_OUT_FIELD_NAME_MAPPING map
                        if(IN_OUT_FIELD_NAME_MAPPING.get(fieldName) != null) {
                                final SolrField field = core.getField(fieldName);
                                        // substitute incoming field name by output field name according to
                                        IN_OUT_FIELD_NAME_MAPPING table
```

```
                                        writeField(out, document, field);
                                }
                        }
                }

                /**
                 * Overridden version of method that substitutes incoming field name by output field name according to
IN_OUT_FIELD_NAME_MAPPING map.
                 *
                 * @param out writer to write XML to
                 * @param document <code>SolrDocument</code> to get field value from
                 * @param field field to write
                 * @throws TransformerException in case of any exception
                 */
                @Override
                protected void writeField(final Writer out, final SolrDocument document, final SolrField field) throws
TransformerException {
                        final String fieldName = field.getName();
                        final SolrFieldType fieldType = field.getType();
                        // create new instance of SolrField based on field argument but with desired name
                        final String newFieldName = IN_OUT_FIELD_NAME_MAPPING.get(fieldName);
                        final SolrField newField = new BaseSolrField(newFieldName, fieldType, field.getDataType());

                        switch(fieldType) {
                                case SIMPLE:
                                case DYNAMIC:
                                        final Object content = document.getFieldValue(fieldName);
                                        writeXMLElement(out, newField, null, content);
                                        break;
                                case MULTIVALUED:
                                case DYNAMIC_MULTIVALUED:
                                        final Collection<Object> values = document.getFieldValues(fieldName);
                                        // in output all values for multivalued field should be wrapped by parent XML
                                        element.
                                        // name of that parent XML element == field name + 's'
                                        final String rootElementForMultivalued =
                                        getXMLElementNameForMultivaluedField(newField);
                                        writeXMLStartTag(out, rootElementForMultivalued, null);
                                        for(final Object value : values) {
                                                writeXMLElement(out, newField, null, value);
                                        }
                                        writeXMLEndTag(out, rootElementForMultivalued);
                                        break;
                                default:
                                        LOG.warn("Unknown field type [{}] was encountered", fieldType);
                        }
                }
}

/**
 * Base implementation of {@link SolrField}. It's supposed that client of this class knows to which core current field
belongs to.
 */
class BaseSolrField implements SolrField {

        private final String name;
        private final String outputName;
        private final SolrFieldType type;
        private final SolrFieldDataType dataType;
        private final String sortName;
        private final String cleanedName;

        /**
         * Creates instance based on outputName, field type, field data type and sort name.
         *
         * @param outputName field name as it is in Rest WS output
         * @param type field type
         * @param dataType field data type
         * @param sortName field name from Solr schema that is used for sorting
         */
        public BaseSolrField(final String outputName, final SolrFieldType type, final SolrFieldDataType dataType, final
String sortName) {
```

```
                this.outputName = outputName;
                this.type = type;
                this.dataType = dataType;
                this.sortName = sortName;
                if(type == SolrFieldType.DYNAMIC || type == SolrFieldType.DYNAMIC_MULTIVALUED) {
                        this.name = outputName.concat("_");
                } else {
                        this.name = outputName;
                }
                this.cleanedName = outputName;
        }

        /**
         * Creates instance based on outputName, field type and field data type.
         *
         * @param outputName field name as it is in Rest WS output
         * @param type field type
         * @param dataType field data type
         */
        public BaseSolrField(final String outputName, final SolrFieldType type, final SolrFieldDataType dataType) {
                this(outputName, type, dataType, null);
        }

        @Override
        public String getName() {
                return name;
        }

        @Override
        public String getOutputName() {
                return outputName;
        }

        @Override
        public SolrFieldType getType() {
                return type;
        }

        @Override
        public SolrFieldDataType getDataType() {
                return dataType;
        }

        @Override
        public String getSortName() {
                return sortName;
        }

        @Override
        public String getCleanedName() {
                return cleanedName;
        }
}
```

Compiling Java-based Custom Output Code

After you have created a new class for your Java-based custom output format, you must compile the class using JDK 1.6.x. To successfully compile you must setup the class path correctly. Use the following libraries and directories that are available within:

`${JBOSS_HOME}/server/${serverName}/deploy/SearchManager.war`.

To compile Java-based custom output code, follow the steps outlined in the Setting up a Workspace section on page 18.

Deploying Java Custom Output Code

After your new output format has been created and compiled, you are ready to register and upload the new plug-in to the system.

To register and deploy a Java-based custom output format:

1. Within the Media Suite User interface, navigate to **Discovery > Transformers > Java Transformers**.
2. Click **Add New**.
3. Add a Format Code

Note The Format Codes for transformers should be unique.

---

4. Upload the configured JAR file.
5. Click **Save and Activate**.

Note If there are XSLT-based and Java-based formats with the same name, the XSLT format will take precedence.

Chapter 10

Customizing Feed Parsers

Overview

Each feed provider makes information available to the EPG module in a slightly different manner by providing different fragment types. For example, one feed provider may publish all information in a single file that can be parsed as a SINGLE fragment type, while others might provide different files for STATIONs, SCHEDULEs, and PROGRAMs respectively.

EPG provides the ability to parse and ingest the following standard fragment types:

- SINGLE
- SCHEDULE
- PROGRAM
- PROGRAM_EXTRA
- STATION
- LINEUP
- REGION

Any fragment type not listed above is considered a custom fragment type and can be created by modifying and extending the default EPG feed parsers. The following chapter explains the process of creating your own feed parser.

Understanding the EPG Data Model

EPG allows you to create custom parsers that give you the flexibility to use feeds from different providers**.** Custom parsers allow you to implement parsing for any standard or custom fragment types as required.

This section provides background information that is needed prior to creating a custom parser. Topics include:

Core Entities

In general, it is the role of any custom parser to convert incoming parsed data into the following set of core entities:

- Station
- Region
- ChannelMap
- Program
- Schedule

All of the listed entities have important mandatory fields that must be populated. See the "EPG Entity Relationship Diagram" on page 163 to see the required and optional fields. The following table consists of the core entities that can be extended with sets of additional metadata.

Table 30 Core Entities that can be Extended

Entity	Extensions Available
station	Set of images, genres and searchable custom fields: • Set `<StationImage>` • Set `<StationGenre>` • Set `<StationCustomField>`
region	None
channel map	None
program	Set of images, searchable custom fields, and additional IDs: • Set `<ProgramImage>` • Set `<ProgramCustomField>` • Set `<ProgramId>` • Set `<ProgramGenre>` • Set <ProgramContributor>
schedule	Set of unsearchable custom fields: • Set `<ScheduleCustomField>`

Custom Fields

Any custom metadata that is not part of the standard EPG data model, can be added for the following entity types:

- `ScheduleCustomField`
- `StationCustomField`
- `ProgramCustomField`

`ScheduleCustomField` is an additional metadata type that can be represented in a name-value pair form. These fields are stored in the index, but are not searchable. Each `ScheduleCustomField` must be initialized with a name and a value. The name and value are both required and cannot be left empty.

`StationCustomField` and `ProgramCustomField` can be set to either searchable or not searchable. To make them searchable, the `SearchFieldType` property must be set to true. False is the default value.

The `SearchFieldType` property must be initialized with one of the following `CustomFieldType` enum values:

- `CustomFieldType.DATE`
- `CustomFieldType.NUMERIC`
- `CustomFieldType.TEXT`

In addition, you will need to specify a custom field name and value.

In the index, custom fields will be given special field names. For custom fields with type:

- `CustomFieldType.DATE`, the index field name is `cf_date_<custom field name>`
- `CustomFieldType.NUMERIC`, the index field name is `cf_num_<custom field name>`
- `CustomFieldType.TEXT`, the index field name is `cf_text_<custom field name>`

The Search Manager API will allow you to use these custom field names in the filter portion of the search query when searching for programs and stations. You will need to specify both the custom field name and the desired value within the searches.

See the provided source code for an example of creating custom fields.

Images

For station and program entities, you can create a list of images.

Table 31 Station and Program images

Image	Common Fields	Additional Fields
`StationImage`	URI, height and width	primaryFlag and defaultFlag
`ProgramImage`	URI, height and width	category (which stores the image type)

Note `StationImage` and `ProgramImage` must be initialized with the URI field. All other fields are optional.

See the provided source code for an example of creating custom images.

Genres

For station and program entities, you can create a list of genres.

Genres may be added in the same manner as shown in the `ParserProgramWrapper`:

```
public void addGenre(final String genre) {
  if(!genres.contains(genre)) {
    genres.add(genre);
    program.getProgramGenres().add(new ProgramGenre(program, new Genre(genre,
    null)));
    }
  }
```

See the provided source code for an example of creating custom genres.

Additional Program IDs

Programs can contain additional program IDs stored as name-value pairs within the database. These program IDs are those that are not present in the core Program entity or can be ones that you would like to make available to the API. Additional program IDs are used during API searches.

The following sample code adds an additional program ID to a program:

```
public void setExternalId(final String externalId) {
    program.setExternalId(externalId);
    program.getProgramIds().add(new ProgramId(program,
    AdditionalIdentifierName.TMS_ID, externalId));
  }
```

Examples of Additional Program IDs

Depending on their external identifiers, programs might have additional tmsid, tmssrseid and tmsseaids assigned to them.

A program with the external ID EP007473660116 would have the following IDs:

Table 32 Program 1 Example

Name	Value
tmsid	EP007473660116
tmssrseid	187511
tmsseaid	188458

A program with the external ID EP000009909467 would have the following IDs:

Table 33 Program 2 Example

Name	Value
tmsid	EP000009909467
tmssrseid	187511
tmsseaid	188459

In Search Manager you can find Program 1 by using the following request:

`/opencase/sm/resource/rest/related?regionId=xxx&id=tmsseaid:equals:188458`

You can find Programs 1 and 2 by using this request:

`/opencase/sm/resource/rest/related?regionId=xxx&id=tmssrseid:equals:187511`

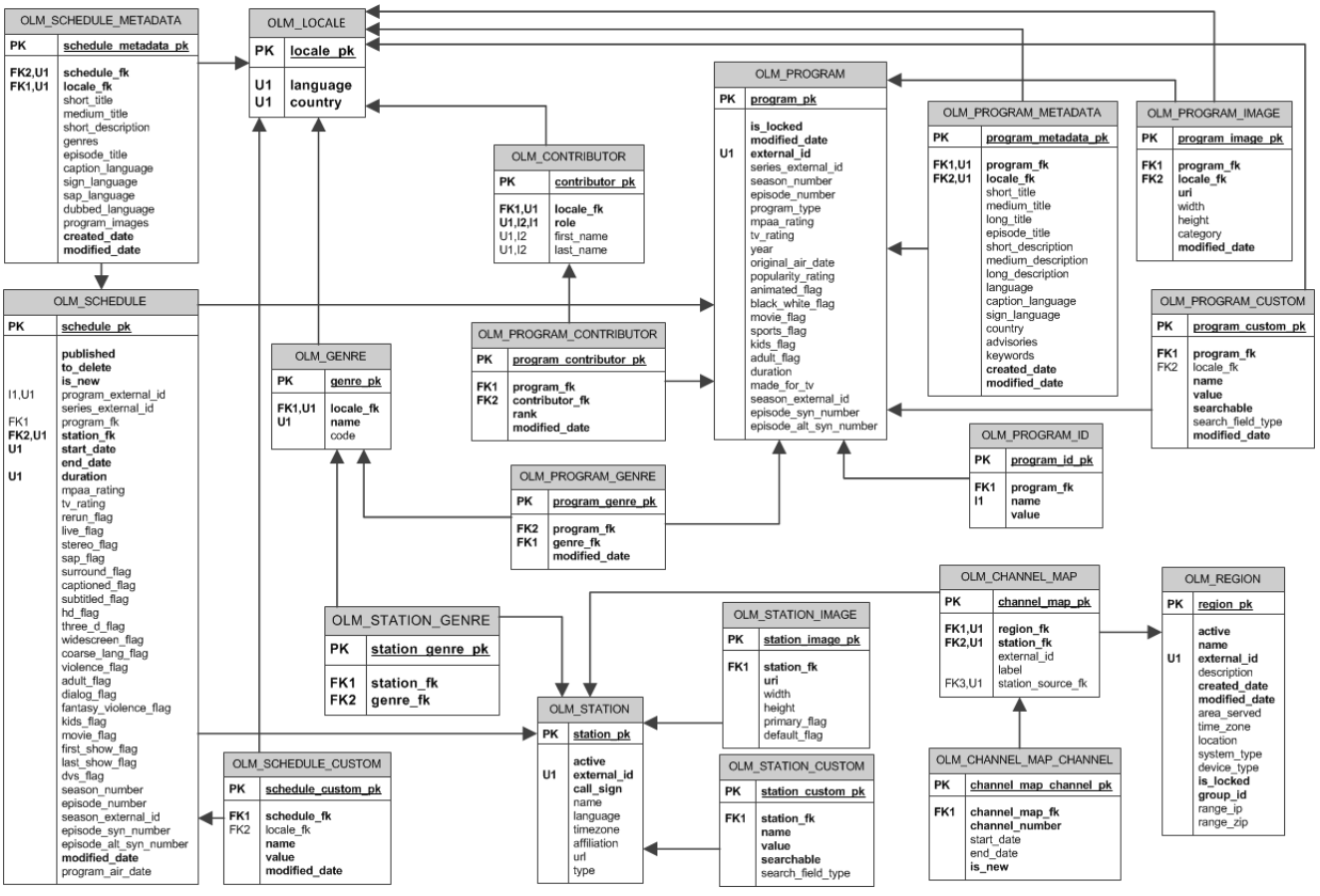
See the provided source code for an example of creating additional program IDs.

Entity Relationship Diagram

The following diagram illustrates the fields and relationships that belong to each entity within the EPG data model. Required fields are shown in bold. The entity names and field names are very similar, which should allow you to easily map the sample database model to the Java model.

Along with the well known PK (Primary Key) and FK (Foreign Key) abbreviations, U1 refers to "unique", and I1 refers to "index".

Figure 9 EPG Entity Relationship Diagram



Creating Custom Parsers

At a minimum, parsers must do the following:

- Parse an input stream into sets of entities. Parsers should not close the input stream as this action will be performed by core EPG logic.
- Pre-process parsed entities if required. This pre-processing can include data adjustments, corrections, call outs to external systems, or similar steps.
- Store parsed data into databases via the provided instance of `FeedProcessorManager`.

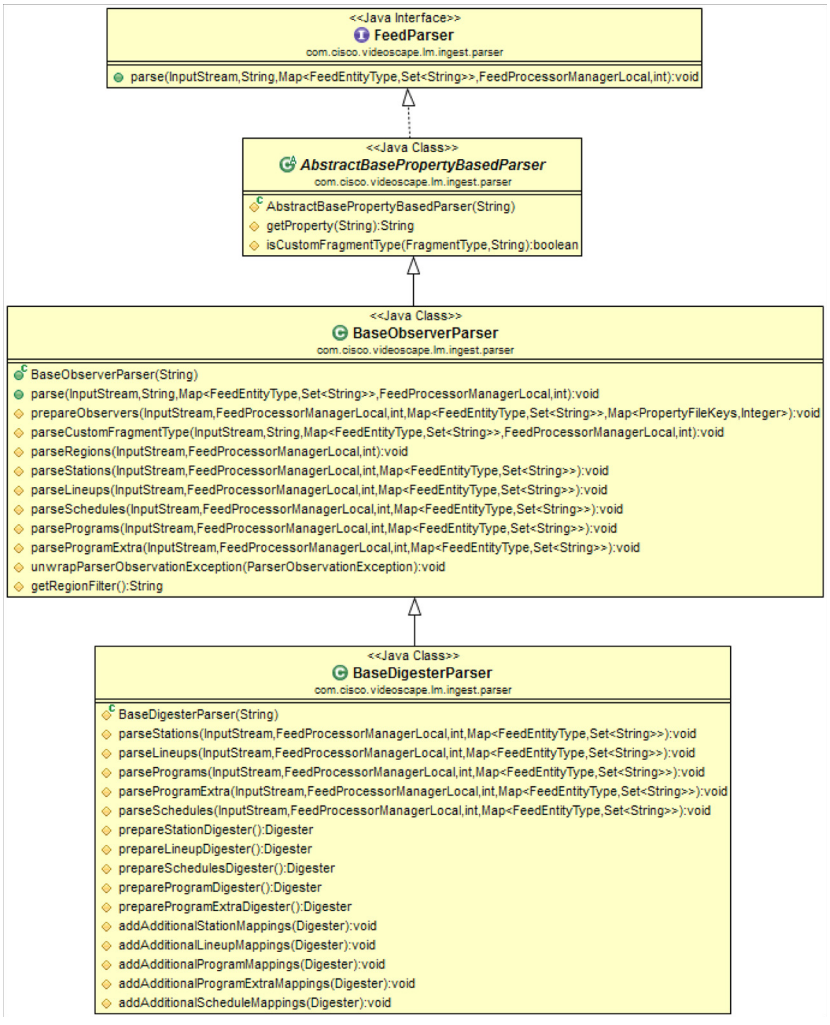
Extending an Existing Parser

A custom parser must implement at least one method in the `FeedParser` interface. At the developer's discretion, existing classes may be extended to take advantage of provided utilities in the parser-related package. The following section describes the existing parser and supporting classes.

Class Diagram

This diagram shows how base parser classes are available for extending the parsing of new feeds.

Figure 10 Parser Class Diagram



Any parser class has to implement the `FeedParser` interface, whether directly or via base classes.

Base Parser Classes

A brief overview of base parser classes is provided in the following table. For more information, please refer to the Javadocs.

Table 34 Base Parser Classes

com.cisco.videoscape.lm.ingest.parser.FeedParser	
	Contains the method that performs parsing of the EPG feed: `void parse(InputStream inputStream, String fragmentType, Map<FeedEntityType, Set<String>> FeedProcessorManagerLocal processor, int batchSize) throws FeedParserException` Parameters: • inputStream – InputStream for EPG content to be read and ingested. • fragmentType – An identifier for the type of fragment this file represents. • processor – Callback interface for the parser to call as a batch is parsed and ready to be ingested. • batchSize – Number of items to parse before calling the feed processor to process the batch. In order to limit resource usage, batch size should be respected. See description of observer pattern below for details on how to implement batch size for digester type parsing.
com.cisco.videoscape.lm.ingest.parser.AbstractBasePropertyBasedParser	
	Can be used as a base class for all parsers. Has the built-in ability to read property files. The property filename has to be passed into the constructor to make properties from the file available to the `getProperty(key)` method. This class also contains the protected method `isCustomFragmentType(FragmentType fragmentType, String originalFragmentType)`. This method is used in all default parsers to check whether a feed fragment type can be handled by the parser. The method has to be overridden in cases where the parser is going to parse a non-standard fragment type so that it returns true for all fragment types supported by the parser.
com.cisco.videoscape.lm.ingest.parser.BaseObserverParser	
	This class extends `AbstractBasePropertyBasedParser` and defines the default implementation for the parse and stub methods for all standard and custom fragment types. This parser uses an observable approach for parsing entities and splitting them into batches. See “Understanding the Observable Parsing Approach” on page 167.

Table 35 Base Parser Classes Continued

com.cisco.videoscape.lm.ingest.parser.BaseDigesterParser	

	This class extends `BaseObserverParser` with support for XML parsing via the Digester library and defines protected methods for digester creation.
	e.g. `protected Digester prepareStationDigester(){}`
	These methods are implemented for all standard fragment types. They create the Digester object, and call the `addAdditionalMappings` method for that fragment type:
	
	`protected Digester prepareStationDigester() {`
	`final Digester digester = new Digester();`
	`digester.setValidating(false);`
	
	`addAdditionalStationMappings(digester);`
	`return digester;`
	`}`
	You can implement parsing for any of the standard fragment types by extending `BaseDigesterParser`. To do so, simply override the `addAdditionalXXXMappings` or `prepareXXXDigester()` methods for the fragment types that your parser will support.
	If you want to override an existing parser implementation (such as GLF or TMS ON) and would like to change base mappings for an entity, then you will need to override this method and rewrite the definitions in the above code sample for the following line of code: `addAdditionalStationMappings(digester);`

Classes Involved in Parsing

The following table provides an overview of the classes involved in the parsing process. For more information, refer to the Javadocs.

Table 36 Classes Involved in Parsing

com.cisco.videoscape.lm.model	
	Package that contains entities created by the parser to be saved into the database. • ChannelMap • Contributor • Genre • Program • ProgramContributor • ProgramGenre • ProgramID • ProgramCustomField • ProgramImage • Region • Schedule • ScheduleCustomField • Station • StationCustomField • StationGenre • StationImage

Table 37 Classes Involved in Parsing Continued

com.cisco.videoscape.lm.model	
	EPG has predefined sets of entities that are wrappers for database entities. The parser usually creates wrapper entities as their structure is closer to the feed data structure. The feed processor manager can store the following types of entities: • Station • Schedule • ChannelMap • Region • Program All other entities not listed above are children of these entities. As a result, any data parsed by the parser must be converted into one of these parent entities.
com.cisco.videoscape.lm.ingest.parser.model	
	Contains classes that wrap database entities for parsing purposes to create a structure that is closer to the feed data model. Wrapper classes provide some preprocessing/conversion logic that transforms string metadata into a format that is compatible with the underlying entity. The following classes are available: • `ParserEntity<T>` – interface that is implemented by all wrapper classes. It contains the single method `T getParsedEntity()`, which returns the database entity represented in the concrete wrapper implementation. • `ParserLineupWrapper` – parser wrapper for the ChannelMap database entity. • `ParserProgramWrapper` – parser wrapper for the Program database entity. • `ParserRegionWrapper` – parser wrapper for the Region database entity. • `ParserScheduleWrapper` – parser wrapper for the Schedule database entity. • `ParserShowCardWrapper` – parser wrapper for the Program database entity used for showcards parsing. • `ParserStationWrapper` – parser wrapper for that Station database entity. Each wrapper provides the ability to retrieve the populated entity expected by the feed processor. At a minimum, this is done by implementing the `ParserEntity` interface.

Understanding the Observable Parsing Approach

All default EPG parsers use an observer-based approach for parsing. With an observable approach, each parsed entity is passed to the `ObservableEPGEntity` class. All of the other work performed by this entity, such as splitting parsed entities into batches and saving those batches is done automatically by the set of observer classes registered in the `ObservableEPGEntity`.

Steps used in an observer-based approach:

1. The parser parses content into a single entity (program, station, etc.).
2. The parser invokes `ObservableEPGEntity.getInstance().notifyObservers(parsedEntity)`.
3. The observer for `parsedEntity`:
   a. Adds the `parsedEntity` into an internal list.
   b. Checks whether the size of the list is the same as the batch size.

c Once a batch is completed, the entire batch is passed into the `FeedProcessorManager` instance to be saved.

d After the batch is saved, the list is cleared to allow for the garbage collection of processed objects as necessary.

Table 38 Observable Parsing Classes

com.cisco.videoscape.lm.ingest.parser.observer.AbstractBaseObserver<T>	
	This class extends `java.util.Observer` and contains a field for holding a list that contains a single batch. The batch size is initialized during observer instantiation. This class also contains an internal method that performs work that splits entities into batches: • `update(final Observable observable, final Object observedObject)` – splits incoming feed data into batches and invokes `saveBatch()` once each batch is complete. • void saveLastBatch() – invokes `saveBatch()` if there is something in the batch Methods that have to be implemented in each observer implementation (for each entity type) are: • `public abstract void addToBatch(Object observedObject);` Implementations of this method should check whether the `observedObject` parameter contains a wrapper entity or whether an entity is ready for saving. In the wrapper entity instance, this method should create a core entity via a call to `getParsedEntity()`. The resulting core entity is then added to the batch. • `public abstract void saveBatch()throws FeedProcessorManagerException;` Implementations of this method are supposed to invoke the feed processor to persist collected batch of entities and clear the batch container variable named "entities".
Observer implementations of the com.cisco.videoscape.lm.ingest.parser.observer	
	The following implemented observer classes are available: • `ProgramObserver` – observer for Program-related entities (Program and ParserProgramWrapper). Performs whitelist-based filtering and does the additional work of choosing titles/descriptions closest to the required length. • `StationObserver` – observer for Station-related entities (Station and ParserStationWrapper). • `ScheduleObserver` – observer for Schedule-related entities (Schedule and ParserScheduleWrapper). Performs whitelist-based filtering. • `LineupObserver` – observer for ChannelMap related entities (ChannelMap and ParserLineupWrapper). Performs whitelist-based filtering. It also creates regions from lineups and performs region name (headend name) filtering. • `RegionObserver` – observer for Region related entities (Region and ParserRegionWrapper). Performs region name (headend name) filtering. • `ShowCardObserver` – observer for ParserShowCardWrapper entities (containing additional program info). Performs whitelist-based filtering.
com.cisco.videoscape.lm.ingest.parser.observer.ObservableEPGEntity	
	Extends `java.util.Observable`. All available observers are registered in this class. This is the main entry point for saving parsed objects. To process an entity after parsing, call the following method: `ObservableEPGEntity.getInstance().notifyObservers(parsedEntity);` which will add the parsed entity to a batch and call `FeedProcessorManager` to save the entity.
com.cisco.videoscape.lm.ingest.parser.observer.ObserversHolder	
	This class contains all registered observers.

Default Parser Implementations

The following section describes the default parsers that are provided with the EPG module.

Default parsers include:

1. `com.cisco.videoscape.lm.ingest.parser.glf.GLFFeedParser`
   This parser is used for the GLF file format and is based upon the observable approach. The GLF data structure contains cross-references to the feed file. Therefore, to build a parsed object, you must have the entire file parsed to memory. This parser can be extended anywhere as required – starting from helper methods like `parsePopularityRating(..)`, `processProgramValues(..)`, etc. and finishing with complete digester preparation.
2. `com.cisco.videoscape.lm.ingest.parser.tmson.TMSOnFeedParser`
   This parser is used for TMS On feeds that are provided in the form of separate XML files (one file per fragment type). The parser uses the observable approach and does not require that the entire file be parsed. Instead, objects are parsed and passed to observers one at a time. This parser can be extended, and parsing can be redefined for each fragment type.
3. `com.cisco.videoscape.lm.ingest.parser.tmson.TMSOnTVScheduleParser`
   This parser is used for TMS TV Schedules feeds that are provided in pipe-delimited files. The parser uses the observable approach and does not require the entire file to be parsed. Instead, objects are parsed and passed to observers one at a time. This parser can be extended, and parsing can be redefined for each fragment type.
4. `com.cisco.videoscape.lm.ingest.parser.tva.TVAFeedParser`
   This parser is used for TVA feeds that are provided in the form of XML files. Only Stations, Schedules and Programs can be parsed. This parser can be extended to either support new fragment types or to redefine existing feed fragment mappings.
5. `com.cisco.videoscape.lm.ingest.parser.tva.RedBeeFeedParser`
   This parser is used for RedBee feeds and is an extension to the TVA parser with additional mappings and program preprocessing. This parser can also be extended.

Compiling the Custom Parser

After you have created the new class for your plug-in, you must compile the class using JDK 1.6.x. Follow the steps outlined in the Prerequisites section of this document for Preparing a Workspace.

Deploying Custom Parsers

After your new parser has been created and compiled, you are ready to deploy the new plugin to the system.

To deploy the custom parser:

1. Navigate to **EPG > Setup > Parser Plugins.**

Note A list of default parser plug-ins is displayed. By default, the GLF Parser is active. Only one parser can be active at a time. All other parsers must be deactivated.

---

2. Click **Add New**.

3 Enter a value in the Configuration field.

4 Upload the `.jar` file.

5 Click **Activate.**

6 Optional: Clean-up previously ingested data when switching EPG providers. This option is available after activating a new parser.

Custom Parser Testing

The best way to check whether your parser works is to deploy it and run the ingestion workflow using your parser. To check the parser code with your tests, you must mock a part of core EPG functionality. The only class that you need to mock is `WorkflowStatusManagerUtil`. Therefore, in the provided source code, the implemented custom parser test sample also mimics `WorkflowStatusManagerUtil`.

The test was written on TestNG framework and mocking is done via JMockit.
See http://testng.org/doc/index.html and http://code.google.com/p/jmockit/ for further details on those tools.

Sample Custom Parser Code

Sample source code related to this chapter is shown in the following section and will also be provided along with this document. See the `Custom Input (Feed Parsing) Files` folder for sample code that is specific to this chapter.

Implementing a New Parser

In this example we will write a parser for the XMLTV format. (For details on this format, see http://wiki.xmltv.org/index.php/XMLTVFormat.) To implement this parser, you will need to extend the `BaseDigesterParser` class. The information in XMLTV format is provided in a single file. This allows you to either implement the parsing of a single fragment type with all entities combined, or implement separate parsing of different fragment types. An example of this would be to use the same file for different fragment types for either a station or schedule. The following samples are created with the assumption that you are implementing a parser using multiple fragment types.

The XMLTV format only provides information for channels and schedules that have program information. You must override, and then implement, the following two methods:

- `addAdditionalStationMappings(..)`
- `addAdditionalScheduleMappings(..)`

A representation of the stations feed XML follows:

Figure 11 Stations Feed XML

```
<tv source-info-url="http://www.schedulesdirect.org/" source-info-name="Schedules
Direct" generator-info-name="XMLTV/$Id: tv_grab_na_dd.in,v 1.70 2008/03/03
15:21:41 rmeden Exp $" generator-info-url="http://www.xmltv.org/">
  <channel id="I10436.labs.zap2it.com">
    <display-name>13 KERA</display-name>
    <display-name>13 KERA TX42822:-</display-name>
    <display-name>13</display-name>
    <display-name>13 KERA fcc</display-name>
    <display-name>KERA</display-name>
    <display-name>KERA</display-name>
```

```
    <display-name>PBS Affiliate</display-name>
    <icon src=http://host/station.logo" />
  </channel>
</tv>
```

To implement station parsing you must override the `addAdditionalStationMappings` method. The override will look similar to the following:

Figure 12 Method Override

```
/**
 * Overriding this method allows you to add mapping to implement a stations parser.
 *
 * @param digester created digester object that you will fill with our mappings.
 */
@Override
protected void addAdditionalStationMappings(final Digester digester) {
  // add mapping to create instance of the Listing entity when parser meets "tv"
  // element
  digester.addObjectCreate("tv",
  com.cisco.videoscape.lm.ingest.parser.observer.Listing.class);
  // configure station parsing
  // add mapping to create instance of the ParserStationWrapper entity when parser
  // meets "tv/channel" element
  digester.addObjectCreate("tv/channel", ParserStationWrapper.class);
  // make parser know that method has to be called to add parsed station to the   /
  // list of all parsed stations when "tv/channel"
  // element is closed. This method is called on the instance of the Listing,
  created when "tv" element was met
  digester.addSetNext("tv/channel", "addStation");
  // map which methods to call when different sub-tags of "tv/channel" are met
  digester.addSetProperties("tv/channel", "id", "externalId");
  digester.addCallMethod("tv/channel/display-name", "setName", 0);
  digester.addCallMethod("tv/channel/display-name", "setCallSign", 0);
  digester.addCallMethod("tv/channel/url", "setUrl", 0);

  // add images parsing
  // make parser create instance of StationImage when "tv/channel/icon" element is
  // met
  digester.addObjectCreate("tv/channel/icon", StationImage.class);
  // define which method to call when "tv/channel/icon" is closed
  digester.addSetNext("tv/channel/icon", "addImage");
  // map tag attributes to entity fields
  digester.addSetProperties("tv/channel/icon", new String[] { "src", "width",
  "height" }, new String[] { "uri", "width", "height" });
  }
```

Since the channel entity structure is very similar to the EPG station entity, you can use the following classes that are already available in EPG:

- `com.cisco.videoscape.lm.ingest.parser.observer.Listing`
- `com.cisco.videoscape.lm.ingest.parser.model.ParserStationWrapper`.

As you can see, the majority of the parsing is performed by the Digester and default EPG classes.

The schedule and program information, however, is more complex because the EPG data structure differs slightly from what is found in XMLTV. As a result, when you are parse schedules you will need to extend/implement several EPG classes.

This following EPG class is used if you are adding schedules to a list of parsed schedules (where each schedule tag is already parsed):

Figure 13 Add Schedules to a List of Parsed Schedules

```
package com.mycompany.parser;

import com.cisco.videoscape.lm.ingest.parser.model.ParserEntity;
import com.cisco.videoscape.lm.ingest.parser.observer.Listing;
import com.cisco.videoscape.lm.ingest.parser.observer.ObservableEPGEntity;

/**
 * Container for parsed entities.
 */
public class XMLTVListing extends Listing {

  /**
   * Add {@link ParserEntity} for processing.
   *
   * @param parsedEntity instance of parsed entity
   */
  public void addEntity(final ParserEntity parsedEntity) {

  ObservableEPGEntity.getInstance().notifyObservers(parsedEntity.getParsedEntity(
  ));
  }
  }
```

To add an XMLTV representation of the schedule entity:

Figure 14 XMLTV Representation of the Schedule Entity

```
package com.mycompany.parser;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.concurrent.TimeUnit;

import org..commons.lang.StringUtils;

import com.cisco.videoscape.lm.ingest.parser.model.ParserEntity;
import com.cisco.videoscape.lm.model.Schedule;
import com.cisco.videoscape.lm.model.Station;

/**
 *  Implementation of ParserEntity with date parsing and general fields mappings
specific for an XML TV Parser.
 */
public class XMLTVSchedule implements ParserEntity<Schedule> {

  private Schedule schedule = new Schedule();


  public void setStartDate(String date) {
    schedule.setStartDate(parseDate(date));

  }
```

```
  public void setEndDate(String date) {
    schedule.setEndDate(parseDate(date));
  }

  private Date parseDate(final String date) {
    SimpleDateFormat sdf = new SimpleDateFormat("yyyyMMddHHmmss Z");
    try {
      Date parsedDate = sdf.parse(date);
      return parsedDate;
    } catch(ParseException e) {
      e.printStackTrace();
    return null;
  }
}

  public void setTitle(String title) {
    schedule.setShortTitle(title);
  }

  public void setDescription(String description) {
    schedule.setShortDescription(description);
}

  public void setEpisodeNumber(String system, String episodeNumber) {
    if("onscreen".equals(system)) {
      schedule.setEpisodeNumber(Integer.valueOf(episodeNumber));
    }
  }

  public void addCategory(String category) {
    if(StringUtils.isBlank(schedule.getGenres())) {
      schedule.setGenres(category);
    } else {
      schedule.setGenres(schedule.getGenres() + "," + category);
  }
}

  public void setStationId(String stationId) {
    final Station station = new Station();
    station.setExternalId(stationId);
    schedule.setStation(station);
  }

  // all other fields can be added here

  public Schedule getParsedEntity() {
    schedule.setDuration((int)TimeUnit.MILLISECONDS.toMinutes(schedule.getEndDate().g
    etTime() - schedule.getStartDate().getTime()));
    return schedule;
  }
}
```

Inside the feed, schedule information will be presented in the following manner:

Figure 15 Schedule Information Within the Feed

```
<tv source-info-url="http://www.schedulesdirect.org/" source-info-name="Schedules
Direct" generator-info-name="XMLTV/$Id: tv_grab_na_dd.in,v 1.70 2008/03/03 15:21:41
rmeden Exp $" generator-info-url="http://www.xmltv.org/">
  <programme start="20080715003000 -0600" stop="20080715010000 -0600"
```

```
   channel="I10436.labs.zap2it.com">
    <title lang="en">NOW on PBS</title>
    <desc lang="en">Jordan's Queen Rania has made job creation a priority to help curb
     the staggering unemployment rates among youths in the Middle East.</desc>
    <date>20080711</date>
    <category lang="en">Newsmagazine</category>
    <category lang="en">Interview</category>
    <category lang="en">Public affairs</category>
    <category lang="en">Series</category>
    <episode-num system="dd_progid">EP01006886.0028</episode-num>
    <episode-num system="onscreen">427</episode-num>
    <audio>
      <stereo>stereo</stereo>
    </audio>
    <previously-shown start="20080711000000" />
    <subtitles type="teletext" />
  </programme>
</tv>
```

To implement schedules parsing you must override the `addAdditionalScheduleMappings` method and define your mappings. The implementation appears as follows:

Figure 16 Override the addAdditionalScheduleMappings method

```
/**
 * Override the base method and define mappings for schedule entity parsing.
 *
 * @param digester created digester object which you will fill with our mappings.
 */
@Override
  protected void addAdditionalScheduleMappings(final Digester digester) {
  // make parser create our custom XMLTVListing object when "tv" element is opened
  digester.addObjectCreate("tv", XMLTVListing.class);
  // configure schedule parsing
  // make parser create our custom XMLTVSchedule object when "tv/programme"
  element is opened
  digester.addObjectCreate("tv/programme", XMLTVSchedule.class);
  // make parser add parsed schedule via addEntity of the XMLTVListing
  digester.addSetNext("tv/programme", "addEntity");
  // map attributes of the tv/programme tag to XMLTVSchedule fields
  digester.addSetProperties("tv/programme", new String[] { "start", "stop",
  "channel" }, new String[] { "startDate", "endDate",
      "stationId" });

  // map sub-elements of the tv/programme tag to XMLTVSchedule fields
  digester.addCallMethod("tv/programme/title", "setTitle", 0);
  digester.addCallMethod("tv/programme/desc", "setDescription", 0);
  digester.addCallMethod("tv/programme/category", "addCategory", 0);
  digester.addCallMethod("tv/programme/episode-num", "setEpisodeNumber", 2);
  digester.addCallParam("tv/programme/episode-num", 0, "system");
  digester.addCallParam("tv/programme/episode-num", 1);
  // add mapping to the to make "review" tag be saved as custom field
  digester.addCallMethod("tv/programme/review", "setReview", 0);
  // all other field mappings go here
}
```

The PROGRAM fragment type parsing implementation should be done in the same manner as for schedules. An implementation is included in the provided source code.

Extending an Existing Parser

The following section assumes that you want to modify program parsing for the `TMSOnFeedParser` to create a limited version of a program with a minimal set of fields. The parser sample will only parse the Program `externalId`, `seriesExternalId`, `episodeTitle`, `episodeSynNumber`, `episodeAltSynNumber` and `title` fields. To implement this functionality, you must extend `TMSOnFeedParser` and override the `prepareProgramDigester()` method. In `prepareProgramDigester` we will only load the digester object that has the required entities.

Figure 17 Extending an Existing Parser

```
package com.mycompany.tmsonextension;

import org..commons.digester.Digester;

import com.cisco.videoscape.lm.ingest.parser.model.ParserProgramWrapper;
import com.cisco.videoscape.lm.ingest.parser.observer.Listing;
import com.cisco.videoscape.lm.ingest.parser.tmson.TMSOnFeedParser;

/**
 * Extension to TMSOnFeedParser which parses only limited program info.
 */
public class MyTMSOnParser extends TMSOnFeedParser {

  @Override
  protected Digester prepareProgramDigester() {
    final Digester digester = new Digester();
    digester.setValidating(false);
    digester.addObjectCreate("on", Listing.class);
    digester.addObjectCreate("on/programs/program", ParserProgramWrapper.class);
    digester.addSetNext("on/programs/program", "addProgram");
    digester.addSetProperties("on/programs/program", new String[] { "TMSId",
    "seriesId" }, new String[] { "externalId", "seriesExternalId" });

    digester.addCallMethod("on/programs/program/episodeInfo/title",
    "setEpisodeTitle", 0);
    digester.addCallMethod("on/programs/program/episodeInfo/synNum",
    "setEpisodeSynNumber", 0);
    digester.addCallMethod("on/programs/program/episodeInfo/altSynNum",
    "setEpisodeAltSynNumber", 0);
    digester.addCallMethod("on/programs/program/titles/title", "addTitle", 3, new
    Class[] { String.class, String.class, Integer.class });
    digester.addCallParam("on/programs/program/titles/title", 0);
    digester.addCallParam("on/programs/program/titles/title", 1, "type");
    digester.addCallParam("on/programs/program/titles/title", 2, "size");
  return digester;
}
}
```

Chapter 11

Custom Components

This chapter includes the following information related to custom components:

Overview

Components are composed of properties, common entities, and custom attributes all of which form the building blocks for bundles. Although Media Suite comes with a default set of components, developers may need to create custom components to suit the unique requirements of a particular deployment. Custom components cannot be created via the Media Suite user interface and must be added programmatically. When created, components are specific to the Media Suite Content Manager and will be managed solely by that application module.

In general, components comprise the following:

- A database table, which defines the component's structure
- A Java class with annotations that enable Media Suite to rent the component dynamically on the user interface and make it available for use in the Content Manager XSD and with SOAP Web Services.
- a text resource file, which defines labels for the interface
- configuration information, which enables you to register the component into Media Suite.

Configuring Your Java IDE

Prior to developing for Media Suite, certain changes must be made to your development environment in order for your Java IDE to properly compile code and to reference appropriate dependencies.

To set up your environment:

1. Using the Java IDE of your choice, create a new project.
2. Go to `Installer\setup\deployable` and copy `ContentManager.ear` to a local folder.
3. Expand the `ContentManager.ear` file.
4. Copy the `compass-2.2.0.jar` file from the `lib` directory into your project and add it to the build path.
5. Copy the `OpenCASECommon-client.jar` file into your project and add it to the build path.
6. Copy the `ContentManager-ejb.jar` file into your project and add it to the build path.
7. Add the file `ejb3-persistence.jar` to your project and add it to the build ptah. This file is NOT included in the ContentManager.ear file and must be copied from the JBoss 5.1 Application Server. The file is located in the $`JBoss_HOME\client\` directory and a version of it is included at `Installer\server\jboss-5.1.0.GA.zip`.

The following procedures are required for creating a custom component:

1. "Creating the Database Table"
2. "Creating the Custom Component Class"
3. "Updating the ComponentExtensionRegistry Class"
4. "Creating Text Resource Files"
5. "Registering the Custom Component"
6. "Deploying the Custom Component"

Creating the Database Table

To begin creating a custom component, you establish the component structure by creating a database table. The following example is written in MS SQL Server. Each database field corresponds to a component property.

Figure 18 Data Definition Language for a Sample Component

```
CREATE TABLE OCM_EXAMPLE_COMPONENT
(
    EXAMPLE_COMPONENT_PK INT NOT NULL,
        EXAMPLE_NAME NVARCHAR(255),
        ENUM_STRING NVARCHAR(255),
        EXAMPLE_NUMBER INT,
        LARGE_TEXT NVARCHAR(4000),
        EXAMPLE_DATE DATETIME,
        DOUBLE_NUMBER NUMERIC(18,5),


    CONSTRAINT OCM_EXAMPLE_COMPONENT_PK PRIMARY KEY(EXAMPLE_COMPONENT_PK),
    CONSTRAINT EXAMPLE_COMPONENT_COMP_FK FOREIGN KEY (EXAMPLE_COMPONENT_PK)
        REFERENCES OCM_COMPONENT (COMPONENT_PK)
);
```

Note Marking fields as "required" is currently not supported in the Media Suite user interface.

Data Type Mappings

The following table lists Java data types with corresponding data types for the two databases supported by Media Suite: MS SQL Server, and Oracle.

Table 39 Data Type Mappings

Java Type	SQL Server	Oracle
String	NVARCHAR(desired size)	VARCHAR2(desired size)
Integer	INT	NUMBER(10,0)
Long	BIGINT	NUMBER(20,0)
Double	NUMBER(18,5)	NUMBER(18,5)
Date	DATETIME	TIMESTAMP
Boolean	TINYINT	NUMBER(3,0)

Creating the Custom Component Class

In this section, we will create a Java class that forms the core of the custom component. The class will contain fields that map to the columns of the table created in the previous section.

Create the Java Class

To create the Java class:

1 Create a component class by writing a custom Java class extending from:

`com.extend.opencase.cm.model.component.AbstractComponent` or one of its child classes.

This class represents a custom component, and is a Java Entity that will map to a table that was created in the database.

Figure 19 Starting a Custom Component Class

```
public class ExampleComponent extends AbstractComponent implements Serializable {

    private static final long serialVersionUID = -6026543024873684482L;
```

Note The example above includes a random UUID. Make sure that you generate a unique version of your own UUID.

Implement Required Methods

Two methods need to be implemented to fully integrate the component with OpenCASE.

To implement the required methods:

1 Add a constant to your new component and call it ENTITY_ALIAS. Set the constant's value to the name of your class (in upper case), using an underscore character to delimit words. For example, if the Java class is called ExampleComponent then add the following:

```
public static final String ENTITY_ALIAS = "EXAMPLE_COMPONENT";
```

2 Implement the method *getEntityAliasValue* as follows.

```
@Transient
public String getEntityAliasValue () {
    return ENTITY_ALIAS;
}
```

3 Implement the method *getComponentLabelKey* as follows.

```
@Transient
public String getComponentLabelKey () {
    return "component.type.examplecomponent";
}
```

The value used in this method will be used in "Creating Text Resource Files when the message properties for the component are created.

Note For both methods, the @Transient annotation is required.

Adding Class-Level Annotations

Next, we will add class-level information in the form of Annotations, that enables Media Suite to render the component in the user interface, use it in relevant web services, and include it within the XML schema for the system.

Steps:

1 Add Java Persistence PI (JPA) annotations. These annotations will define how the component maps to the database.

Figure 20 Java Class with JPA Annotations

```
@Entity
@DiscriminatorValue (value=ExampleComponent.ENTITY_ALIAS)
@Table (name="OCM_EXAMPLE_COMPONENT")
@PrimaryKeyJoinColumn (name="EXAMPLE_COMPONENT_PK",referencedColumnName = "COMPONENT_PK")

public class ExampleComponent extends AbstractComponent implements Serializable {

    private static final long SerialVersionUID = -6026543024873684482L;
```

2 Add Compass Annotations. These annotations enable full text search capabilities in the Media Suite user interface and within web services.

Figure 21 Java Class with Compass Annotations

```
@Entity
@DiscriminatorValue(value = ExampleComponent.ENTITY_ALIAS)
@Table(name = "OCM_EXAMPLE_COMPONENT")
@PrimaryKeyJoinColumn(name = "EXAMPLE_COMPONENT_PK", referencedColumnName = "COMPONENT_PK")
@PrimaryKeyJoinColumn
@Searchable (alias = ExampleCompoent.ENTITY_ALIAS, root = true, poly = true, extend =
AbstractComponent.ENTITY_ALIAS)

public class ExampleComponent extends AbstractComponent implements Serializable {

    private static final long serialVersionUID = -6026543024873684482L;
```

3 Add JAXB Annotations. These annotations enable the component's use in the Content Manager XSD as well as the Content Manager SOAP web services.

Figure 22 Java Class with JAXB Annotations

```
@Entity
@DiscriminatorValue(value = ExampleComponent.ENTITY_ALIAS)
@Table(name = "OCM_EXAMPLE_COMPONENT")
@PrimaryKeyJoinColumn(name = "EXAMPLE_COMPONENT_PK", referencedColumnName = "COMPONENT_PK")
@PrimaryKeyJoinColumn
@Searchable(alias = ExampleCompoent.ENTITY_ALIAS, root = true, poly = true, extend = Abstract-
Component.ENTITY_ALIAS)
@XmlType (name = "ExampleComponent", namespace = XMLConstant.CM_NAMESPACE)

public class ExampleComponent extends AbstractComponent implements Serializable {

    private static final long serialVersionUID = -6026543024873684482L;
```

4 Add the Media Suite Annotation. This annotation is required so that the component can be dynamically rendered within the Media Suite user interface.

Figure 23 Java Class with Media Suite Annotations

```
@Entity
@DiscriminatorValue(value = ExampleComponent.ENTITY_ALIAS)
@Table(name = "OCM_EXAMPLE_COMPONENT")
@PrimaryKeyJoinColumn(name = "EXAMPLE_COMPONENT_PK", referencedColumnName = "COMPONENT_PK")
@PrimaryKeyJoinColumn
@Searchable(alias = ExampleCompoent.ENTITY_ALIAS, root = true, poly = true, extend = Abstract-
Component.ENTITY_ALIAS)
@GroupSorting (groups = { ExampleComponent.FIELDS_GROUP_1, ExampleComponent.FIELDS_GROUP_2, Exam-
pleComponent.FIELDS_GROUP_3 })
@XmlType(name = "ExampleComponent", namespace = XMLConstant.CM_NAMESPACE)

public class ExampleComponent extends AbstractComponent implements Serializable

    private static final long serialVersionUID = -6026543024873684482L;
```

Adding Properties to the Component

For each column in the database, a number of tasks must be performed:

1 Add a property to the Java class with the appropriate data type. For information on appropriate data types, See "Data Type Mappings". For the EXAMPLE_NAME column in the sample database table, we would add the following:

```
private String exampleName;
```

2 Add getter and setter methods for the property. For example:

```
public String getExampleName () {
    return exampleName;
}

public void setString (String string) {
    this.exampleName = string;
}
```

3 Add required JPA annotations to the getter method. For most methods, the @Column annotation is the only one that is required. For example:

```
@Column (name = "EXAMPLE_NAME", nullable = true, length = 225)
public String getExampleName () {
    return exampleName;
}
```

Note Date objects will require an extra JPA annotation. Example: @Temporal (TemporalType.TIMESTAMP)

---

4 Add required Compass annotations to the getter method. For example:

```
@Column(name = "EXAMPLE_NAME", nullable = true, length = 225)
@SearchableProperty (name = "exampleName")
public String getExampleName () {
    return exampleName;
}
```

5 Add required JAXB annotations to the getter method. For example:

```
@Column(name = "EXAMPLE_NAME", nullable = true, length = 225)
@SearchableProperty(name = "exampleName")
@XmlElement (name = "exampleName", namespace = XMLConstant.CM_NAMESPACE, required = false)
public String getExampleName () {
    return exampleName;
}
```

6 Media Suite annotations control how the component appears on the form. For additional details on available options, see "Annotations".

Add required Media Suite annotations to the getter method. For example:

```
@Column(name = "EXAMPLE_NAME", nullable = true, length = 225)
@SearchableProperty(name = "exampleName")
@XmlElement(name = "exampleName", namespace = XMLConstants.CM_NAMESPACE, required = false)
@FormElement (labelKey = "component.example.component.exampleName", //
orderNumber = 1,//
groupKey = ExampleComponent.FIELDS_GROUP_1,//
bulkEditable = true)
public String getexampleName () {
    return exampleName;
}
```

Sample Java Code

The following sample Java code shows the complete component class with all anotations included for our sample table. this example shows all of the simple data types. It shows a String (enumString) that will be rendered as a drop down UI control. it also shows the use of a Common Entity (categories). Common Entities are normalized global values that are intended for reuse within specific components. Components can use any common entity without requiring a corresponding database column in its own table.

Figure 24 Final Sample Java Code

```
package com.extend.opencase.cm.model.ExampleComponent;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.Iterator;
import java.util.List;
import java.util.Set;

import javax.persistence.Column;
import javax.persistence.DiscriminatorValue;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.PrimaryKeyJoinColumn;
import javax.persistence.Table;
import javax.persistence.Transient;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlElementWrapper;
import javax.xml.bind.annotation.XmlTransient;
import javax.xml.bind.annotation.XmlType;

import org.compass.annotations.Searchable;

import com.extend.opencase.cm.dynamicform.annotation.CommonEntityFormElement;
import com.extend.opencase.cm.model.ComponentConstants;
import com.extend.opencase.cm.model.Locale;
import com.extend.opencase.cm.model.XMLConstants;
import com.extend.opencase.cm.model.commonentity.Category;
import com.extend.opencase.cm.model.component.AbstractComponent;
import com.extend.opencase.occ.client.dynamicform.annotation.FormElement;
import com.extend.opencase.occ.client.dynamicform.annotation.GroupSorting;
import com.extend.opencase.occ.client.validator.MaxLengthValidator;

/**
 * The Example Custom Component entity
 *
 */
@Entity
@DiscriminatorValue(value = ExampleComponent.ENTITY_ALIAS)
@Table(name = "OCM_EXAMPLE_COMPONENT")
@PrimaryKeyJoinColumn(name = "EXAMPLE_COMPONENT_PK")
@GroupSorting(groups = { ComponentConstants.ABSTRACT_COMPONENT,
ExampleComponent.GROUP_LABEL_KEY })
@Searchable(alias = ExampleComponent.ENTITY_ALIAS, root = true, poly = true, extend =
AbstractComponent.ENTITY_ALIAS)
@XmlType(name = "ExampleComponent", namespace = XMLConstants.CM_NAMESPACE)
public class ExampleComponent extends AbstractComponent implements Serializable {
```

```
private static final long serialVersionUID = 2779562096054139994F;

private static final String COMPONENT_LABEL_KEY = "component.example.component";
   public static final String GROUP_LABEL_KEY = "component.exampleComponent";

private Locale locale;
       private String exampleName;
       private String url;
   private Set<Category> categories = new HashSet<Category>(0);

   public static final String ENTITY_ALIAS = "EXAMPLE_COMPONENT";

       /**
        * Constructor for Example Custom Component class
        */
       public ExampleComponent() {
              super();
   }

/**
        * Gets Locale of title
        *
        * @return localeFK Locale of title
        */
       @ManyToOne(fetch = FetchType.LAZY)
       @JoinColumn(name = "LOCALE_FK", nullable = true)
       @FormElement(labelKey = "component.exampleComponent.locale", //
       orderNumber = 1, //
       groupKey = GROUP_LABEL_KEY, //
       bulkEditable = true)
       @XmlElement(name = "locale", namespace = XMLConstants.CM_NAMESPACE, required =
false)
       public Locale getLocale() {
              return locale;
       }
   }

       /**
        * Sets Locale of title
        *
        * @param localeFK Locale of title
        */
       public void setLocale(Locale locale) {
              this.locale = locale;
   }

/**
        * Gets exampleName for Example Custom Component in provided locale
        *
        * @return exampleName exampleName for Example Custom Component
```

```
         */
        @Column(name = "ExampleName", nullable = true, length = 255)
        @FormElement(labelKey = "component.exampleComponent.exampleName", //
        orderNumber = 2, //
        groupKey = GROUP_LABEL_KEY, //
        bulkEditable = true)
        @XmlElement(name = "exampleName", namespace = XMLConstants.CM_NAMESPACE,
required = false)
        @MaxLengthValidator(max = 255)
        public String getExampleName() {
               return exampleName;
    }

/**
         * Sets exampleName for the example custom component in provided locale
         *
         * @param exampleName exampleName
         */
        public void setExampleName(String exampleName) {
               this.exampleName = exampleName;
        }

        /**
         * Gets URL for the example custom component
         *
         * @return url URL of example custom component
         */
        @Column(name = "URL", nullable = true, length = 512)
        @FormElement(labelKey = "component.exampleComponent.url", //
        orderNumber = 3, //
        groupKey = GROUP_LABEL_KEY, //
        bulkEditable = true)
        @XmlElement(name = "url", namespace = XMLConstants.CM_NAMESPACE, required =
false)
        @MaxLengthValidator(max = 512)
        public String getUrl() {
               return url;
    }

/**
         * Sets URL of the example custom component
         *
         * @param url URL of the example custom component
         */
        public void setUrl(String url) {
               this.url = url;
    }

/**
         * Get the set of categories
         *
```

```
         * @return Set of categories
         */
        @CommonEntityFormElement( //
        labelKey = "commonentity.type.category", //
        orderNumber = 4, //
        groupKey = GROUP_LABEL_KEY, queryFilters = {})
        @XmlElement(name = "category", namespace = XMLConstants.CM_NAMESPACE)
        @XmlElementWrapper(name = "categories", namespace = XMLConstants.CM_NAMESPACE)
        @Transient
        public Set<Category> getCategories() {
               return this.categories;
    }

@XmlTransient
        @Transient
        public List<Category> getCategoriesAsList() {
               List<Category> categoryList = new ArrayList<Category>(categories);
               return categoryList;
    }

        /**
         * Set the set of categories
         *
         * @param Categorys Set of categories
         */
        @Transient
        public void setCategories(Set<Category> categories) {
               this.categories = categories;
    }

/**
         * Add a category
         *
         * @param Category A category
         */
        @Transient
        public void addCategory(final Category category) {
               this.categories.add(category);
    }

/**
         * Remove a category
         *
         * @param category A category
         */
        @Transient
        public void removeCategory(final Category category) {
               removeCommonEntity(category, this.categories);
    }

/**
```

```
         * Remove all categories
         */
        @Transient
        public void removeAllCategories() {
               Iterator<Category> iter = this.categories.iterator();
               while(iter.hasNext()) {
                      Category category = iter.next();
                      iter.remove();
                      this.removeCategory(category);
               }
    }

/**
         * {@inheritDoc}
         *
         * @return {@code EXAMPLE_COMPONENT}
         */
        @Override
        @Transient
        public String getEntityAliasValue() {
               return ENTITY_ALIAS;
    }

/**
         * {@inheritDoc}
         */
        @Override
        @Transient
        public String getComponentLabelKey() {
               return COMPONENT_LABEL_KEY;
        }

    }
```

Updating the ComponentExtensionRegistry Class

The ComponentExtensionRegistry class is used to ensure that newly added components are available for use in the SOAP APIs.

To update the ComponentExtensionRegistry class:

1. Create a new file called ComponentExtensionRegistry.java.
2. Copy the following code into the class:

```
package com.extend.opencase.cm.model.component;

import javax.xml.bind.annotation.XmlSeeAlso;
import javax.xml.bind.annotation.XmlTransient;

@XmlTransient
public class ComponentExtensionRegistry {
```

```
    public ComponentExtensionRegistry () {
        super ();
    }
}
```

3 Add the following text after the @XmlTransient line to add the new component:

```
@XmlSeeAlso ({ExampleComponent.class})
```

4 Compile the class.

5 Replace the existing class located in the com/extend/opencase/cm/model/component folder of the ContentManager-EJB module of the deployed application. Ensure that the original package and directory structure is maintained.

Creating Text Resource Files

To enable a component to be properly rendered in the user interface, its properties must be mapped to entries in a text resource file for each language that you wish to support. This is done by creating an entry for each property using the labelKey specified in its FormElement. To create a resource file, you must create key/value pairs as shown in the following example:

Figure 25 Sample Component Text Resource File

```
## Component name
component.type.examplecomponent = Example Component

## Field groups
component.fieldgroup.1=Field Group A
component.fieldgroup.2=Field Group B
component.fieldgroup.2=Field Group C

## Fields
component.example.component.exampleName=Example Name Label
component.example.component.exampleNumber=Example Number Label

component.example component.doubleNumber=Double Number Label
component.example.component.largeText=Large Text Label
component.example.commonentity.type.category=Category Label
component.example.component.enumString=Enumerated String Label

#Enum Values
INSERT INTO OCM_RESOURCE_KEY (RESOURCE_KEY_PK, RESOURCE_KEY) VALUES (10004,
'component.example.component.enumString.type.valuea');
INSERT INTO OCM_RESOURCE_LABEL (RESOURCE_LABEL_PK, RESOURCE_LABEL, RESOURCE_KEY_FK, LANGUAGE)
VALUES (10004, 'Value A', 10004, 'en-US');

INSERT INTO OCM_RESOUCE_KEY (RESOURCE_KEY_PK, RESOURCE_KEY) VALUES (10005,
'component.example.component.enumString.type.valueb');
INSERT INTO OCM_RESOURCE_LABEL (RESOURCE_LABEL_PK, RESOURCE_LABEL, RESOURCE_KEY_FK, LANGUAGE)
VALUES (10005, 'Value B', 10005, 'en_US');

INSERT INTO OCM_RESOURCE_KEY (RESOURCE_KEY_PK, RESOURCE_KEY) VALUES (10006,
'component.example.component.enumString.type.valuec');
```

```
INSERT INTO OCM_RESOURCE_LABEL (RESOURCE_LABEL_PK, RESOURCE_LABEL, RESOURCE_KEY_FK, LANGUAGE)
VALUES (10006, 'Value C', 10006, 'en_US');
```

This text file should be saved as <name>_<locale>.propeties. For each language that an administrator use interface will be used in, a separate properties file should be created.

For example:

example-component_en.properties

and

example-component_fr.properties

The text resource file should then be placed in the following folder:

`ContentManager.ear/ContentManager-web.war/WEB-INF/classes`

A reference to this file must be added to `ContentManager-web/WEB-INF/components.xml` that is show below. New text is bolded in this example:

Figure 26 Referencing the Resource File in components.xml

```
<component name="transportBundles"
class="com.extend.opencase.occ.client.resourceloader.ResourceBundle"
scope="application" startup="true">
            <property name="bundleNames">
                     <key>messages</key><value>messages</value>
                     <key>menu</key><value>menu</value>
                     <key>loadfromDB</key>
                     <value>com.extend.opencase.cm.model.ResourceLabel</value>
                     <key>example-component</key><value>example-component</value>
            </property>
</component>
```

Note The key above needs to match the main part of the text resource filename. In this case, "example-component" must match "**example-component**_en.properties".

Registering the Custom Component

To register the newly created component with Media Suite, SQL statements must be written that populate various tables with specific component information. These statements make the system aware of the component. The following section will explain the statements that must be executed to register your custom component.

To register a custom component:

1 The following statement created an entry in the OCM_RESOURCE_KEY table that corresponds to the new component's label key. This was specified in step three of "Implement Required Methods".

Table 40 Create Resource Key

Statement:	`INSERT INTO OCM_RESOURCE_KEY (RESOURCE_KEY_PK, RESOURCE_KEY) VALUES (<resourceKeyPk>, <resourceKey>);`
Parameters:	

resourceKeyPk	A valid primary key for this table
resourceKey	A String containing the label key for this component. This must correspond to the value used in the getComponentLabelKey method.

Example:

```
INSERT INTO OCM_RESOURCE_KEY (RESOURCE_KEY_PK, RESOURCE_KEY)
VALUES (76, 'component.type.examplecomponent');
```

2 The following SQL statements register this component with the system. They also provide the ability to add a reference to an image file that will appear in the user interface.

Table 41 Add to Type Table

Statement:

```
INSERT INTO OCM_TYPE (TYPE_PK, NAME, DISCRIMINATOR,
RESOURCE_KEY_FK, SUB_TYPE, IMAGE) VALUES (<typePk>, <name>,
<discriminator>, <resourceKeyFk>, <subType>, <image>);
```

Parameters:

typePk	A valid primary key for this table.
name	A String with the component name. The value must match the @XmlType annotation in the Java class.
discriminator	A String containing the exact value, "COMPONENT_TYPE".
resourceKeyFk	The resourceKeyPk specified in Step 1.
subType	A String containing the exact value, "SYSTEM".
image	The name of an image file that will be used to represent this component in the Media Suite user interface.

Example:

```
INSERT INTO OCM_TYPE (TYPE_PK, NAME, DISCRIMINATOR,
RESOURCE_KEY_FK, SUB_TYPE, IMAGE) VALUES (74, 'ExampleComponent',
'COMPONENT_TYPE', 76, 'SYSTEM', 'image.gif');
```

Table 42 Add Component to Type Table

Statement:

```
INSERT INTO OCM_COMPONENT_TYPE (COMPONENT_TYPE_PK,
IMPLEMENTATION_CLASS, COMPONENT_DISCRIMINATOR) VALUES
(<componentTypePk>, <implementationClass>,
<componentDiscriminator>);
```

Parameters:

componentTypePk	The type primary key.
implementationClass	A String containing the fully qualified Java classname of the component.
componentDiscriminator	The ENTITY_ALIAS of this component as specified in step one of "Implement Required Methods".

Example:

```
INSERT INTO OCM_COMPONENT_TYPE (COMPONENT-TYPE_PK,
IMPLEMENTATION_CLASS, COMPONENT_DISCRIMINATOR) VALUES (75,
'com.extend.opencase.cm.model.component.ExampleComponent',
'EXAMPLE_COMPONENT');
```

Deploying the Custom Component

Deploying a component will allow it to be recognized and available in Media Suite. This is the final stage for creating a custom component.

The following steps deploy a component:

1. Add the component information to Compas.
2. Deploy the Java classes.
3. Deploy the text resource file(s).

Adding the Component to Compass

Compass is utilized in Media Suite to provide a simple, yet powerful API for performing searches using the Lucene search engine. In order to recognized a new component, Compass' configuration file, named compass.cfg.xml, must be updated to include a <class> tag entry for each new component. The Compass configuration file is located at:

`ContentManager.ear/ContentManager.sar/compass.cfg.xml`

The bolded text in the following excerpt from the configuration file show where <class> tags should be added:

Figure 27 Compass Configuration File

```
<compass-core-config xmlns="http://www.compass-project.org/schema/core-config"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.compass-
project.org/schema/core-config http://www.compass-project.org/schema/compass-core-config-
2.2.xsd">
<compass name="ContentManagerCompass">
    <!-- The location of the index (Lucene Directory) -->
    <connection>
        <file path="indexes/ocm"/>
    </connection>

    <!-- Class mappings -->
    <mappings>
        <!-- Because compass does not have VFS support, we have to add the whole classpath. -->
        <class name-"com.extend.opencase.cm.model.commonentity.AbtractCommonEntity"/>
        <class name="com.extend.opencase.cm.model.commonentity.Genre"/>
        <class name="com.extend.opencase.cm.model.commenentity.Advisory"/>
        <class name="com.extend.opencase.cm.model.commonentity.Category'/>
        <class name="com.extend.opencase.cm.model.commonentity.Rating"/>
        <class name="com.extend.opencase.cm.model.commonentity.Style"/>
        <class name="com.extend.opencase.cm.model.commonentity.Talent"/>
        <class name="com.extend.opencase.cm.model.commonentity.TalentRole"/>
        <class name="com.extend.opencase.cm.model.commonentity.Person"/>
        <class name="com.extend.opencase.cm.model.component.ExampleComponent"/>
```

The Compass configuration file, `compass.cfg.xml`, may be found in the:

`ContentManager.ear/ContentManager.sar` folder.

Deploying the Java Classes

Copy the component Java class into the following folder:

```
ContentManager.ear/ContentManager-ejb.jar/{PATH OF THE PACKAGE}
```

For example:

```
ContentManager.ear/ContentManager-ejb.jar/com/extend/opencase/cm/model/component
```

Copy the ComponentExtensionRegistry.class into the following folder:

```
ContentManager.ear/ContentManager-ejb.jar/com/extend/opencase/cm/model/component
```

Deploying the Text Resource Files

All component text resource files should be placed in the following location on the server:

```
ContentManager.ear/ContentManager-web.war/WEB-INF/classes
```

Deploying the Image File

The image file that represents the component within the Media Suite user interface should be copied to the following location:

```
ContentManager.ear/ContentManager-web/img/componenticons
```

Verifying the Component

Congratulations, you have created your custom component! Now is a good time to verify that the component exists in Media Suite in the way that you inteded.

To verify a component:

1. Create and use the component with the Media Suite user interface.
2. Validate that the component appears within the Content Manager XSD.
3. Validate that the component appears within the Content Manager Component WSDL.

Validating Components with the User Interface

To use the new component within the Media Suite user interface:

1. Navigate to Metadata > Components.
2. Click Add New > [Component Type]. Your new component should appear in the drop-down.
3. Type a Name.
4. Click Create & Edit.
5. Fill out all available fields. The fields will vary based on the characteristics of the component you have created. Ensure that all fields and drop-downs appear and behave as expected.
6. Click Save. Your component should be created.

To verify the fields and contents of a component:

1. Navigate to Metadata > Components to examine the newly created component.
2. Perform a search by entering part of the name followed by an asterisk.
3. Click Search.

4 Click the underlines name of the component. The component details should appear.

For further information on creating or searching for components within the Media Suite user interface, refer to the *Media Suite User Guide*.

XSD Validation

To validate that you component appears within the Content Manager XSD:

1 Go to http://{server}/ContentManager/resource/rest/schema/contentManager.xsd

2 Search for the name of the component as specified in the `@XmlType` annotation. This can be found in the section, "Adding Class-Level Annotations. Search for an `xs:complexType` entry.

3 For each property that was marked with an `@XmlElement`, there should be a corresponding `xs:element` entry.

WSDL Validation

To validate that your component appears within the Content Manager WSDL:

1 Go to http://{server}/ContentManager/webservices/component-service?wsdl

2 Search for the component name as specified in the `@XmlType` annotation. This can be found in the section, "Adding Class-Level Annotations". Search for an `xs:complexType` entry.

3 For each property that was marked with an `@XmlElement`, there should be a corresponding `xs:element` entry.

Note If you are using the Content Manager WS Client JAR that is included with the application in custom code, such as a custom ESB service of Entitlement plug-in, your new component will not be usable even once it appears in the WSDL. If this is required, you will need to create your own client JAR based on the new WSDL using the tool of your choice.

Using the Component

Since components form the building blocks of bundles, the next logical step after successfully creating a custom component would be to create one or more bundles that use this component. If this required by your deployment, this brief section will provide you with some background information on creating bundles.

To use your new component, create new bundles types by either creating a new bundle template or by modifying a copy of an existing template within the Media Suite user interface.

Part of the bundle template creation process is to drag and drop components into various locations in a bundle. After you have created your new component, it should be

Index

L

M

O

P

R

T

U

W