LIFECYCLE MANAGER

LIBRARY CONFIGURATION GUIDE
v6.5

Trademarks

Copyright

Overview

The Lifecycle Manager Library Configuration feature allows customers to easily customize Asset governance processes and extend Lifecycle Manager's standard behavior through an event driven extension mechanism. Library configuration is managed through an XML document known as the Library Process Configuration (LPC) document that defines the structure of Lifecycle Manager's event-driven processes and additionally defines extended functionality that has been "plugged-in" to the Lifecycle Manager product. The Library Configuration engine is built on the Lifecycle Manager Event Notification Framework. As such, it allows customers to define "Listeners" that encapsulate custom behavior and specify the events and conditions that will cause this behavior to be invoked on a per-library basis.

This document provides the details of the structure of the LPC document. However, users are encouraged to use the Eclipse based Configuration Designer Tool to make changes and configure the library.

Notification Concepts

Events

Events represent the occurrence of some state change within Lifecycle Manager. This state change may be the result of automatic or user driven processing. Lifecycle Manager pre-defines a set of Events for significant state changes within the core product. A list of pre-defined Events and their properties may be obtained by executing the command *DisplayEventDefinitions* from the Lifecycle Manager admin console. These types of Events will often contain context information in common fixed attributes or keyed properties specific to the Event. In addition to pre-defined Lifecycle Manager Events, Lifecycle Manager also supports custom Events introduced in the <custom-events> element of the LPC document. For example:

```
<custom-events>
      <custom-event>ASSET_VALIDATED</custom-event>
</custom-events>
```

Each kind of event is uniquely identified by its "event type" (the name of the event) and is additionally classified by the following attributes[1]:

- Category – a grouping for similar events
- Component – the Lifecycle Manager component that the Event pertains to ("ASSETSOURCE" or "LIBRARY")
- Severity – Used to distinguish between Events representing messages, warnings or errors[2].
- Asset ID – the id of the asset this Event pertains to
- Group Name – the name of the organizational group that this Event pertains to
- User ID – the account name of the User that this event pertains to

[1] These attributes are initially populated on pre-defined Lifecycle Manager events when the event is created. Attributes and properties are transferred from triggering events to result events declared in actions.
[2] Severities listed in order of increasing severity are: "INFO_SECONDARY", "INFO_PRIMARY", "WARNING" and "SEVERE".

Events may be raised explicitly by external integrators through the *notifyListeners()* method on the Lifecycle Manager Library API SOAP interface[3]. Events may also be created as the result of the execution of a listener.

Listeners

Listeners are associated with classes or services that encapsulate custom behavior to be invoked upon the occurrence of an Event. Lifecycle Manager supports both internal and external listeners. Lifecycle Manager provides a growing library of internal Listener classes to perform such tasks as UDDI publishing and XML validation. External listeners are those provided by integrators as a web service that Lifecycle Manager will invoke as necessary. Listeners are defined using a "listener" element within the "listeners" section of the LPC document. Each listener defined is given a unique name within the LPC document or containing Library configuration element and is associated with a Listener class. The listener definition may also include properties to use in configuring the listener class instance. The following is an example of a listener definition that defines a listener named "WSDLValidator" that will use the internal listener class "XMLArtifactValidator". A value is specified for the property "target-artifact-category" that will be used to configure the listener instance.

```
<listener name="WSDLValidator" class="XMLArtifactValidator">
      <properties>
      <property name="target-artifact-category" value="wsdl" />
   </properties>
</listener>
```

A list of internal listeners currently available appears in Appendix A.

External Listeners

External listeners are web services that implement Lifecycle Manager's ExternalListener service interface definition[4]. An external listener is defined in similar fashion in the LPC document as an internal listener. However, the "class" attribute will always be set to "ExternalSOAPListener". The properties for an external listener will always include the access point URL of the web service implementing the ExternalListener interface and, optionally, a user ID and password to use for basic HTTP authentication to the service. Any additional properties will be passed through to the external listener service and used for configuration specific to that listener implementation. Here is an example of an external listener definition:

```
<listener name="MyExternalListener" class="ExternalSOAPListener">
      <properties>
            <property name= "accesspoint_url"
              value="http://myserver/services/ExternalListenerSoap" />
            <property name= "auth_userid" value="admin" />
            <property name= "auth_password" value="xyz123" />
            <property name= "my_property" value="foo" />
   </properties>
</listener>
```

[3] See appendix E for additional details.
[4] This interface definition may be found in the External Listener Sample EAR available on the Support Site.

For additional details on implementing external listeners download the External Listener Sample application from the Lifecycle Manager support site.

Property Encryption
It is possible that sensitive data such as passwords may need to be stored in listener properties. To facilitate this, the "property" element on a listener accepts an optional "encrypt" attribute. Specifying "true" for this attribute indicates that the value of the property should be encrypted. Here is an example:

```
<property name="password" value="thepassword" encrypt="true" />
```

The next time the LPC is retrieved, the property value will appear in encrypted form and will have the attribute "encrypted" set to "true" as in the following example:

```
<property name="password" value="9qd3R6h7qec*" encrypted="true" />
```

The "encrypted" attribute is for internal use and indicates to the LPC parser that the property value is in encrypted form.

Filters

Filters encapsulate criteria to be applied to an Event or set of Events. It allows more precise triggering of listener behavior than simply keying off Events. For example, a Filter can be used to specify not only a particular asset Event, but also names of Classification Criteria Sets or Asset Filters that assets must comply with before a listener is notified, allowing listeners to be invoked only for changes to assets of a certain classification. Filters also allow Event severities, categories, User IDs, Asset IDs, Property values[5], and Group names[6] to be added to the filter criteria.
Example:

```
<filter name="WebServiceArtifactUpdates">
   <event>ARTIFACT_CREATED</event>
   <event>ARTIFACT_UPDATED</event>
   <classification-criteria-sets>
     <classification-criteria-set-name>
     WebserviceAssets
     </classification-criteria-set-name>
   </classification-criteria-sets>
 </filter>
```

It is also possible to specify the complement of the specified Classification Criteria Sets in a Filter by specifying the Boolean attribute "complement" as true. This will allow all Events pertaining to assets that *do not* match the specified CCS.
Example:

```
<classification-criteria-sets complement="true">
   <classification-criteria-set-
name>DatabaseApplicableAssets</classification-criteria-set-name>
</classification-criteria-sets>
```

[5] See Custom Properties under Advanced LPC Concepts
[6] In the case of specifying groups in a Filter (using a <groups> sub-element), the <group> element may be used to specify a particular group while the <group-tree> element may be used to specify a group and its decendants.

Similarly, the complement of the set of specified Groups, Asset Filters or Property Filters can be defined by using the "complement" attribute on the appropriate filter sub-element. The following example allows Events for all groups except "group1" and "group2".

```
<groups complement="true">
   <group-name>group1</group-name>
   <group-name>group2</group-name>
</groups>
```

Asset Filters

Asset Filters are similar in structure and purpose to Classification Criteria Sets but are defined within the LPC document itself in the "asset-filters" element. Asset Filters may be used within a Filter element in the same fashion as a CCS, but have the advantage of being defined directly by the LPC author and are immutable from within the Lifecycle Manager application. This is an example of an AssetFilter definition that specifies two particular asset-types:

```
<asset-filters>
   <asset-filter name="WebserviceAssets">
      <classifier-criteria classifier-category="asset-type">
         <value-set>
            <value>Web service</value>
            <value>Web service Deployment</value>
         </value-set>
      </classifier-criteria>
   </asset-filter>
</asset-filters>
```

Asset Filters consist of one or more "classifier-criteria" elements that specify acceptable values for a particular classifier. The above example utilizes a "value-set" sub-element to specify a number of allowed values for an enumerated style classifier ("asset-type" in this case). Classifier-criteria elements may contain different sub-elements for criteria specification depending on the type of the classifier being used:

- "value-set"
  Used to specify a set of accepted values for an enum type classifier. See above example.
- "decimal-range"
  Used to specify a range of values for a decimal type classifier. Example:

```
<decimal-range>
<min>1.0</min>
<max>5.0</max>
</decimal-range>
```

- "date-range"
  Used to specify a range of dates for a date type classifier. Example:

```
<date-range>
<min>2001-01-01</min>
<max>2004-01-01</max>
</date-range>
```

- "boolean-value"
  Used to specify a true or false value for a Boolean type classifier. Example:

```
<boolean-value>
<value>true</value>
</boolean-value>
```

AssetFilters for Asset Types

Lifecycle Manager provides a set of "built-in" asset-filters for filtering on each of the defined asset-type values in the GDT. These filters have a well known naming format and do not have to be explicitly defined in the LPC document. The format of the asset-type filter names is: "asset-type:<*asset_type*>" where "<*asset-type*>" is a particular asset-type value. For example, the asset-filter named "asset-type:Service" will allow only assets of asset-type "Service".

Actions

Actions combine triggering Events and Filters with Listeners and optionally define result Events to be raised when the listener execution is complete. Actions take a list of one or more triggering Events or Filters, an optional Listener, and optional result Events. If multiple triggering Events or Filters are specified the Listener will be triggered if any Event matching the list of specified Events or Filters occurs[7]. Result Events may optionally be associated with a listener return value (a "result-condition"), allowing different Events to be triggered based on the success or failure of the listener's processing[8]. If a Listener is not specified for an Action but a result Event is, the Action simply raises the specified result Event, essentially converting the trigger Events or Filters into the result Event. It should be noted that the attributes and properties of the triggering Event will be passed onto the result Events.

Example:

```
<action name="ValidateArtifacts">
      <trigger-event>
         <event>ASSET_SUBMISSION_Asset Owner_APPROVED</event>
      </trigger-event>
      <listener>WSDLValidator</listener>
      <result-event event="ASSET_SUBMISSION_APPROVED">
            <result-condition>0</result-condition>
      </result-event>
</action>
```

Delayed Actions

Optionally, the firing of a Listener associated with an Action may be delayed by a fixed amount of time through use of the "delay" attribute. This attribute allows a time delay to be specified in minutes on the Action[9]. For example, the Listener named in the following Action definition will run one day after the triggering Event has occurred:

```
<action name="Send Survey" delay="1440">
      <trigger-event>
         <event>ASSET_REGISTERED</event>
      </trigger-event>
      <listener>SendSurvey</listener>
</action>
```

[7] It is also possible to indicate that all specified trigger events (Events or Filters) must have occurred before the specified Listener is invoked. This is done by specifying the attribute "type" as "SYNCHRONIZED" on the action element.

[8] A result-condition of "-1" is used to indicate the case where the listener invocation resulted in an exception.

[9] Please note that delays are currently accurate only to within a ten minutes of the specified delay interval. For example, a delay specified for sixty minutes could result in an actual delay between sixty and seventy minutes.

Timers

Timers allow the triggering of a specified Event repeatedly at a specified interval. Timers are useful for triggering "batch" style processes that may run hourly, nightly, etc. Timers are specified in the "Timers" element of the LPC document as in the following example:

```
<timer name="ReportTimer">
   <interval>10080</interval>
   <event>GENERATE_REPORT</event>
</timer>
```

In this example, a simple Timer is defined that will trigger the custom Event "GENERATE_REPORT" at a weekly interval (10,080 minutes)[10]. The initial interval will start with the setting of the LPC document. The following example introduces some additional advanced concepts:

```
<timer name="TestTimer1">
   <interval>10080</interval>
   <event>GENERATE_REPORT</event>
   <start-time>2005-12-01T01:00:00</start-time>
   <total-fires>12</total-fires>
   <firing-window>60</firing-window>
</timer>
```

The "start-time" attribute is used to designate a starting time for the initial timer interval. If this time is in the past, the first interval will considered to have started at that time and may result in the Timer firing soon after the LPC document is updated if the initial interval has already expired. In this example the start time is set to 1:00 AM on December 1st.

The "total-fires" attribute is used to set the number of times that the Timer will fire its associated event. Once the Timer has fired the designated number of times it will become dormant until reconfigured in the LPC document. When "total-fires" is not specified, as in the previous example, the Timer will continue to fire indefinitely.

The "firing-window" attribute is used to ensure that the Timer only fires within a specified time range of the designated firing time calculated from the start time and the specified interval. The "firing-window" attribute is useful to ensure that a particular process only runs at a specified time of day or week regardless of server outages that might otherwise push the execution to another time.

Role Management

Custom Group Roles

Lifecycle Manager allows custom roles to be defined between Users and Groups (in addition to the standard roles that Lifecycle Manager already supports). This allows customized processes and views to utilize roles not anticipated by Lifecycle Manager. An example would be a representative from the performance team that must approve assets prior to their publication. Custom Group roles are defined in the "group-roles" element in the LPC document. Example:

```
   <group-roles>
            <group-role>PerformanceRep</group-role>
```

[10] As with Action delay intervals, Timer intervals are currently accurate only to within ten minutes.

```
      <group-role>SecurityArchitect</group-role>
      <group-role>DatabaseArchitect</group-role>
   </group-roles>
```

Defining a Group role in the LPC document will make it available in the list of roles to assign to a User in the Group detail and User detail pages of the Lifecycle Manager product.
Group roles pre-defined in Lifecycle Manager are exactly as shown:

- Project Manager
- ACE
- Publisher
- Asset Owner

Role-based Asset Views

Lifecycle Manager allows the presentation of asset data for viewing of a published asset or editing of a catalog asset to be customized based on a user's group roles. This is facilitated in the LPC document with two types of elements: "metadata-groups" and "metadata-views".

Metadata-groups

Metadata-groups provide an LPC author with a means of designating certain asset metadata elements (classifiers, artifacts, relationships) as part of a named group that can be associated with a particular role-based view of the asset. Metadata-group elements are defined within a "metadata-groups" element as in the following example:

```
<metadata-groups>
   <metadata-group name="Design Elements">
      <classifier name="asset-type" />
      <classifier name="intended-domain" />
      <artifact category="design-model" />
      <artifact category="architecture-document" />
      <relationship name="related-pattern" />
   </metadata-group>
</metadata-groups>
```

The names and categories of the asset metadata elements specified correspond to their definitions in the Global Definition Template for the library. The exception to this is the virtual metadata element "USER_DEFINED". This value "USER_DEFINED" may be used within a classifier, artifact or relationship element to generically specify all user-defined classifiers, artifacts or relationships respectively.

ALL_ELEMENTS Metadata Group
For convenience, a "virtual" metadata-group with the name "ALL_ELEMENTS" is provided by Lifecycle Manager. This metadata-group represents all metadata elements currently defined in the Global Definition Template for the library.

Metadata-views

Metadata-views are a set of metadata-groups that make up a particular view of an asset's data. Metadata-views may reference zero or more metadata-groups by name and may optionally specify the complement of any metadata-group, meaning that all elements specified in the GDT

that are *not* in the specified metadata-group will be included in the view. Metadata-view elements are defined within a "metadata-views" element as in the following example:

```
<metadata-views>
   <metadata-view name="Default">
      <metadata-group name="ALL_ELEMENTS" />
   </metadata-view>
      <metadata-view name="Analysis View">
      <metadata-group name="Analysis Elements" />
      <metadata-group name="Requirement Elements" />
      </metadata-view>
</metadata-views>
```

In this example, the "Default" metadata-view is defined to contain all asset elements in the GDT, where the "Analysis View" contains the elements defined in the "Analysis Elements" and "Requirements Elements" metadata-groups.

The following example defines the "Default" group using the "complement" attribute:

```
<metadata-view name="Default">
      <metadata-group name="Restricted Elements" complement="true" />
</metadata-view>
```

In this example the "Default" view will contain all elements in the GDT except those specified in the "Restricted Elements" metadata-group.

Associating Metadata Views with Group Roles

Metadata views may be associated with group roles within the "group-role" element through use of the "metadata-view" sub-element as in the following example:

```
<group-roles default-metadata-view="Default"
             default-edit-metadata-view="EditDefault">
   <group-role name="Analyst">
      <metadata-view name="Analysis View" scope="VIEW"/>
   </group-role>
   <group-role name="Architect">
      <metadata-view name="Analysis View" scope="VIEW"/>          <metadata-
view name="Design View" scope="VIEW"/>
   </group-role>
   <group-role name="Project Manager">
      <metadata-view name="Project Manager View" scope="ALL"/>
   </group-role>
</group-roles>
```

Group-roles may reference zero or more metadata-views. The "metadata-view" sub-element contains a "scope" attribute that indicates whether to view is applicable to viewing published assets, editing catalog assets, or both view and edit. The choices for the scope attribute are "VIEW", "EDIT", or "ALL" with "VIEW" being the default. Group-roles not referencing any metadata-views will implicitly be associated with the metadata-view specified in the "default-metadata-view" and "default-edit-metadata-view" attributes of the "group-roles" element[11].

[11] For the sake of reverse compatibility, the default-metadata-view attribute is optional in the LPC schema. If this attribute is not specified it will be implicitly set to the metadata-view named "Default". If the "Default" metadata-view is not explicitly defined, it will be implicitly defined to reference the virtual metadata-group

These attributes also determines the visible meta-data elements for library ad catalog Assets displayed from the Lifecycle Manager remote ("thick") clients.

Ordering

On the Lifecycle Manager web ("thin") client, Asset metadata elements will be displayed in their respective sections (classifiers, artifacts, related assets) of the library asset detail page according to the order they are defined in the metadata-group element[12]. In the case where multiple metadata-groups are referenced for a selected role, the aggregated ordering of elements will be affected first by the order of metadata-views within the group-role, then by metadata-group associations within a metadata-view, and finally by the element ordering within each metadata-group. Elements will only be displayed when they initially appear in the ordering; repeated elements will be ignored.

Ordering of metadata elements on Lifecycle Manager remote ("thick") clients will continue to be alphabetical in all cases.

Default Ordering

The metadata-groups element contains the optional attribute "default-ordering". This attribute determines the ordering of asset metadata elements in the following cases:

- Ordering of elements within the virtual "ALL_ELEMENTS" metadata-group
- Ordering of elements in the complement of a metadata-group

There are two choices for the value of the "default-ordering" attribute:

- "ALPHA" – Indicates that elements should be ordered alphabetically
- "GDT" - Indicates that elements should be ordered according to their definition in the Global Definition Template

Process Concepts

Process Integration Points

SOA has designated certain user actions in the Lifecycle Manager product as extension points for inserting request/approval-style processes. These are known as "process integration points". Enabling a particular process integration point will direct the user to a "submit request" page where they are given the opportunity to create and submit an Asset Request, see the Asset Requests section below. The submission of this request triggers an Event of the form *<process integration point>_REQUESTED* that can be used to kick off an approval process that ultimately leads to the requested action being performed or rejected. The following process integration points are currently defined in Lifecycle Manager:

- "ASSET_SUBMISSION"
  This is the point in the catalog at which a user submits an Asset for publishing.

"ALL_ELEMENTS". The case with the default-edit-metadata-view attribute is slightly different: if not explicitly defined, it will default to showing all metadata elements regardless of the definition of the "Default" metadata-view.

[12] Note that the classifier "asset-type" is an exception to this rule. It will always be present in the asset detail page and will be ordered prior to all other classifiers.

- "ASSET_DELETION"
  This is the point in the catalog at which at user clicks on the "delete" link from the Asset Edit page or through the thick-clients.
- "ASSET_ACQUISITION"
  This process is invoked when a user requests acquisition of a published asset.
- "ASSET_REVOCATION"
  This process is invoked when a user requests the revocation of an asset acquisition.

Process integration points are enabled in the "enabled-processes" element of the LPC document as in the following example:

```
<enabled-processes>
   <process>
      <name>ASSET_DELETION</name>
   </process>
</enabled-processes>
```

Process Filters

Filters similar to those associated with Actions can be introduced for use in finer-grain tailoring of the enablement of a process integration point relative to the context User, Asset, or Group. Process Filters are defined in-line as a "filter" sub-element within the "process" element as in the following example:

```
<process>
   <name>ASSET_DELETION</name>
   <filter>
      <groups>
         <group-name>Group 1</group-name>
         <group-name>Group 2</group-name>
      </groups>
      <classification-criteria-sets complement="true">
         <classification-criteria-set-name>Web Service
Assets</classification-criteria-set-name>
      </classification-criteria-sets>
   </filter>
</process>
```

This example shows the enablement of the ASSET_DELETION process being determined by the context Group and asset type. Specifically, it defines the ASSET_DELETION process as being enabled (meaning that the user will see a request submission page) only for Groups "Group 1" and "Group 2" for assets that are *not* web services.

Sub-elements of the process "filter" element are "groups", "classification-criteria-sets", "asset-filters", "property-filters" and "users". These elements are each optional and are defined using the same syntax as corresponding elements in a top-level "filter" element as described in the Filters section above.

Multiple "filter" elements may be specified within a "process" element. By default, if the criteria of any filter for a process are met, the process will be triggered. Optionally the "required" attribute on the "filter" element may be used to require that the criteria of a filter are met. This is done by specifying a value of "true" for the "required" attribute. This allows for

logical "AND" capability rather than the standard "OR" behavior of non-required filters[13]. The default value for the "required" attribute is "false".

Optional Process Attributes

A "process" element can be further modified with the following optional attributes:

- "request-submission-comment"
  Specifying a value of "false" for this attribute will hide the request submission page from the requester in the case where no "initial-property-constraint-set" element is specified[14]. The default value for this attribute is "true"
- "id"
  A value specified for the "id" attribute will be placed in a property on the resulting request with the name "LL:PROCESS_ID". Often it is useful to use this property for additional filtering of actions related to requests resulting from this process. Such filtering can be done using "property-filter" elements. If not specified, the "LL:PROCESS_ID" property will not be present on the request.

Process Definition Elements

As a syntactical convenience, the LPC schema defines a first level child element called "process-definition" that is useful for grouping LPC elements for a particular process. This element contains elements similar to those contained by the LPC root element, allowing elements related to a particular process to be grouped together into a process-definition and separated from elements supporting other processes. The default LPC document utilizes process-definition elements to separate each of the four main process definitions. Here is an example of a process-definition element from the default LPC encapsulating the asset deletion process:

```
<process-definition name="Asset Deletion" >
      <listeners>
            <listener name="AssetDeletion"
                     class="AssetDeletionListener"></listener>
   </listeners>
      <actions>
      <action name="Delete Asset">
         <trigger-event>
            <event>ASSET_DELETION_APPROVED</event>
         </trigger-event>
         <listener>AssetDeletion</listener>
      </action>
   </actions>
</process-definition>
```

Resolution of references to named elements within a process-definition element (references from Actions to Filters and Listeners) is attempted first to matching elements within the same process-definition and secondly to elements in the global section of the LPC document (those not within the scope of another process-definition element). In the above example, when the "Delete Asset" Action reference to the "Asset Deletion" Listener is resolved it will bind to the Listener defined in this process-definition element regardless of whether another similarly named Listener existed

[13] Note that it makes sense to use either all "required" filters or all "non-required" filters for a process element; mixing the two means that the "non-required" filters (the OR-ed filters) will be ignored.

[14] See Property Constraints section of Appendix I for additional details on the "initial-property-constraint-set" element.

in another process-definition or in the global section of the document. However, if the "Asset Deletion" Listener had not been defined in this process-definition, the parser would attempt to find a similarly named Listener in the global scope, and if not found there, would indicate an error on upload of the LPC document.

Activities

Activities correspond to customizable action links/buttons that are presented to the Lifecycle Manager user at certain points in the application. The availability of these links/buttons is controlled through the LPC document by enabling activities within the "enabled-activities" element. Currently the defined activities are[15]:

- "ASSET_ACQUISITION"
  This activity is associated with the asset acquisition process and controls the presence of the "Acquire" link or button on an asset detail page/window.
- "ASSET_REVOCATION"
  This activity is associated with the asset revocation process and controls the presence of the "Revoke" link or button on an asset detail page/window.

Activities may specify filtering criteria within an optional "filter" element. The filtering criteria may include:

- Specific Groups for which the activity is applicable
- Asset Filters for specifying asset classifier criteria
- Property Filters for specifying required property values[16]
- Role Filters (described in the following section)

Role Filters

Role Filters are defined in the "role-filters" element and contain a set of accepted roles as in the following example:

```
<role-filters>
   <role-filter name="RevocationAuthorizedRoles">
      <role-name name="Asset Owner" asset-owning-group="true" />
      <role-name name="Project Manager" />
   </role-filter>
 </role-filters>
```

The Role Filter in this example specifies that a user must have *either* the "Asset Owner" role on the Asset's owning Group *or* "Project Manager" role on the currently active Project for the filter to be satisfied. The "asset-owning-group" attribute is used to specify whether the role is required on the context Asset's owning Group (a value of "true") or on the event's context Group (a value of "false" or not specified). When associated with an activity, a role filter can be used to control the visibility of an activity link or button based on a user's roles for the active Project and context Asset.

Activity Example

[15] Note that these activities only affect acquisition/revocation from the published asset page, they do not effect acquisition within the asset edit process. Asset-filters associated with acquisition relationships are the recommended mechanism to provide filtering of target asset types.
[16] See Custom Properties under Advanced LPC Concepts

The following example is from the default process definition for revocation:

```
<enabled-activities>
      <activity>
      <name>ASSET_REVOCATION</name>
      <filter>
         <role-filters>
   <role-filter-name>RevocationAuthorizedRoles</role-filter-name>
         </role-filters>
      </filter>
   </activity>
</enabled-activities>
```

This example enables the revocation activity (the "Revoke" link in the asset detail page) only for users that have a role specified in the "RevocationAuthorizedRoles" role filter defined in the previous example.

Asset Requests

Asset Requests are persistent entities that are created and initialized by Lifecycle Manager users when a particular Process Integration Point has been activated. They represent the ongoing state of an asset request as it traverses through the stages of its approval process. An Asset Request may contain the following information:

- Attributes that identify the specific context of this request: Asset ID, Group Name, type of request, and the initiator of the request
- The name of the current state of the request.
- A collection of User notes
- A history of what has occurred for this request
- A list of "approval instances" each containing a required Group role, a state ("pending", "approved", or "rejected") and a User Id if the approval instance is in either the "approved" or "rejected" state
- A collection of properties whose meaning is specific to the process.
- An "active" flag indicating whether the request represents an active process or exists to store historical data about a process that has completed.

The current state, the active status, and the pending approvers of an Asset Request are updated by Listeners involved in the approval process. A common process for an Asset Request is as follows:

1. A User creates an Asset Request by activating a Process Integration Point.
2. A Listener is triggered by the Asset Request submission and notifies some set of users of a particular role for the Group specified in the Asset Request.
3. These users either approve or reject the Asset Request
4. A Listener is triggered by the approver's action to either reject the Asset Request or move it to the next state, possibly resulting in the notification of users in another Group role.
5. Upon reaching the final approval state of the Asset Request, a Listener is triggered that performs the requested action (e.g. asset deletion). At this point the Asset Request is marked as inactive.

The Lifecycle Manager UI provides an Asset Request approval page to facilitate step 3 above. Lifecycle Manager also provides an internal Listener class called "GenericRequestHandler" that can be used to configure standard processes as just described[17].

Asset Request Approval and Rejection Events

In addition to the pre-defined and customer-defined Events described in the Events topic above, there are a special class of Events associated with approval and rejection of an Asset Request. The act of a User approving an Asset Request will generate an Event of the form:

<*request type*>_<*Group role*>_APPROVED

For example:

ASSET_DELETION_Asset Owner_APPROVED

Similarly, the act of rejecting an Asset Request will generate an Event of the form:

<*request type*>_<*Group role*>_REJECTED

For example:

"ASSET_SUBMISSION_Asset Owner_REJECTED

These Events play a significant role in configuring Asset Request state transitions[18].

Advanced Configuration Concepts

Custom Properties

Lifecycle Manager allows custom information to be associated with primary business entities such as Groups, Users and Assets in the form of key/value pairs of strings known as "properties". Through the use of properties it is possible to add information to a context object (generally an asset) that may affect the selection and behavior of future actions in a process. In this case, properties may be thought of as "routing" information attached to a particular asset.

To facilitate the use of properties in customizing Lifecycle Manager behavior, the LPC schema allows properties to optionally be associated with Actions and included in Filters as described in the following sections.

Setting Properties from Actions

An action may optionally specify that properties be added or removed from context objects upon completion of the action. This is done by specifying optional "set-property" or "remove-property" elements in within an action element. Here is an example of an action that operates on the properties of the context catalog asset:

```
<action name="Process Asset Submission">
      <trigger-event>
      <event>ASSET_SUBMISSION_APPROVED</event>
   </trigger-event>
   <listener>CustomListener</listener>
   <set-property name="property 1"
                    value="new value"
                    target="CATALOG_ASSET" />
   <remove-property name="property 2" target="CATALOG_ASSET" />
</action>
```

[17] Additional detail on the "GenericRequestHandler" class is available in Appendix A.

[18] See appendixes C and D for examples of using these events to trigger state transitions.

In this example, the action will add “property 1” with a value of “new value” to the catalog asset and remove the property “property 2”.
Valid “target” context objects for properties consist of the following:

- CATALOG_ASSET
  An asset in the catalog. This target object is relevant for catalog events such as asset creation, updates, and submission.
- LIBRARY_ASSET
  The published version of an asset present in a library. This target object is relevant for library events such as asset acquisition.
- GROUP
  The context organizational group for the event (the “active” Project). This target is relevant for most asset related events in the library.
- USER
  The user that triggered the event. This target is relevant for user driven events in the catalog and library.

Promotion of Asset Properties

When an Asset in the catalog is published to the library, custom properties from the catalog Asset will be copied to the published Asset. Note that existing published Asset properties that are not also present on the Catalog Asset will be preserved.

Using Properties in Filters

A “property-filter” allows acceptable property values for various properties to be specified in a manner similar to Asset Filters. Property-filters are defined within a “property-filters” element as in the following example:

```
<property-filters>
      <property-filter
            name="AcquisitionProjectManagerApprovalRequired">
      <property-criteria
                  property-key="PROJECTMANAGER_APPROVAL_REQUIRED"
                  target="GROUP">
         <property-values>
            <value>true</value>
         </property-values>
      </property-criteria>
   </property-filter>
</property-filters>
```

In the above example, the Property Filter specifies that the property named “PROJECTMANAGER_APPROVAL_REQUIRED” must be present on the Event’s context Group and have a value of “true”.

Once defined, Property Filters may be used when filtering actions, processes or activities.

Shared Properties

Shared properties are a type of custom property that are set directly in the LPC document and shared by all instances of the target business entity. Shared properties are immutable from the

standpoint of the Lifecycle Manager application and can only be changed or removed through updates to the LPC. While customers may defined their own shared properties, the default LPC uses shared properties to set metrics report URLs on Groups and Assets.

Property values support the concept of replacement parameters. These parameters are specified in the form "{parameter}" within the property value. Such parameters are resolved from the context object when property is retrieved. Some replacement parameters currently supported are:

- {library.base_url} : Base part of library URL. For example: http://LifecycleManagerServer.
- {library.id} : Library Id. For example: "167:9".
- {library_asset.urn} : URN of the target library (published) asset.
- {group.urn} : URN of the target Group

Asset Request Properties

Custom property behavior is further enhanced for Asset Requests by allowing properties and their characteristics to be defined and constrained within the LPC document. Appendix I addresses Request Property configuration in detail.

Default Asset Filter

The LPC schema allows for the setting of a "default-asset-filter" element. This optional element can be set to the name of an asset-filter defined in the "asset-filters" element. If specified, the designated asset-filter is implicitly applied to all standard queries run in the library. This mechanism is useful in blocking certain types of assets from appearing in standard searches. The "default-asset-filter" element is a direct child of the LPC root element and is specified as follows:

```
<default-asset-filter>AssetFilter1</default-asset-filter>
```

Artifact Sources

Artifact Sources are used to retrieve artifacts from systems other than Lifecycle Manager that do not support URI based access for artifacts. Artifact Sources are classes that contain custom artifact retrieval logic utilized when a user requests an Artifact from an Asset. The responsibilities of an Artifact Source are quite simple: given reference information for a particular artifact, the Artifact Source instance must return a stream that represents the contents of the requested artifact. How the artifact is actually retrieved is left to each Artifact Source class.

Defining an ArtifactSource

ArtifactSources are defined within the "artifact-sources" element in the global section of the LPC document. Like Listeners, Artifact Source instances are configured with properties that the instance may use in retrieving artifact contents from an underlying system of record. An "artifact-source" element contains a unique name, an implementation class name, and properties specific to that class. Here is an example definition:

```
<artifact-source name="p4"
                 class="com.logiclibrary.artifact.sources.HTTPSource">
   <properties>
```

```
   <property name="url"
                value="http://build:5500/@md=d%26c=Mjq@//depot/" />
   <property name="user" value="admin" />
   <property name="password" value="apassword" encrypt="true" />
   </properties>
</artifact-source>
```

In this example, an Artifact Source named "p4" is defined that utilizes the HTTPSource class. The required properties for the HTTPSource class: "url", "user", and "password" are specified[19].

Utilizing Artifact Sources in Assets

Artifact references in Assets that utilize Artifact Sources are entered in "URL" ("by-reference") form. The reference must be entered in the following format:

```
soa://<Artifact Source Name>/<reference info>
```

Where <Artifact Source Name> is the name assigned to the Artifact Source in the LPC document and <reference info> is any identifying information for the artifact that will be passed through to the Artifact source instance. Here is an example reference using the Artifact Source defined in the earlier example:

```
soa://p4/common/main/services/CurrencyExchange.wsdl
```

In this example, the Artifact Source instance named "p4" will be invoked and provided with the reference information "common/main/services/CurrencyExchange.wsdl". The Artifact Source instance will use this reference information as well as the configuration information from its properties to retrieve the artifact contents from the system of record.

Artifact Transforms

Artifact Transforms are responsible for performing transformations on the contents of an artifact before it is presented to a Lifecycle Manager user. Like Artifact Sources, Artifact Transforms are defined with a unique name, an implementing class, and configuration properties as in this example:

```
<artifact-transforms>
   <artifact-transform name="HTML Transform" class="XSLTTransform">
            <properties>
         <property name="transform-definition"
                        value="http://myserver/transforms/t1.xslt" />
      </properties>
   </artifact-transform>
</artifact-transforms>
```

Once defined, Artifact Transforms can be associated with an Artifact Source as in this example:

```
<artifact-source name="p4"
                 class="com.logiclibrary.artifact.sources.HTTPSource">
   <properties>
   <property name="url"
                value="http://build:5500/@md=d%26c=Mjq@//depot/" />
   <property name="user" value="admin" />
   <property name="password" value="apassword" encrypt="true" />
   </properties>
   <artifact-transform-name>HTML Transform</artifact-transform-name>
```

[19] Note the use of the "encrypt" flag for "password". This concept is defined in Property Encryption sub-section of the Listeners section of this document.

```
</artifact-source>
```

Artifact Transforms are applied to the data from the Artifact Source in the order that they are defined in the "artifact-source" element. The resulting transformed data stream will be presented to the user retrieving the artifact.

Artifact Comparators

Artifact Comparators are classes that contain custom artifact comparison logic utilized when displaying differences for two versions of an artifact. An Artifact Comparator must be able to determine first if differences exist between artifact versions and if so, produce those differences in a document that can be presented to the user. Artifact Comparators are configured and selected based on the file extension of target artifacts.

Defining an Artifact Comparator

Artifact Comparators are defined within the "artifact-comparators" element in the global section of the LPC document. Like Listeners, Artifact Comparator instances are configured with properties that the underlying Artifact Comparator instance may use in performing the comparison operation. An "artifact-comparator" element contains a unique name, an implementation class name, and properties specific to that class. Additionally, the artifact content types (file extensions) that the artifact-comparator is applicable for are listed using <content-type> elements. The content-types specified may not be used by another artifact-comparator. An artifact-comparator instance will be selected when the content type of a changed artifact is declared by the artifact-comparator definition. In the case where the content type (extension) of an artifact has changed between versions, an artifact-comparator will only be selected if it declares both the content types involved.
Here is an example definition:

```
<artifact-comparator name="XMLComparator"
  class="com.soa.repositorymgr.extensions.SampleXMLArtifactComparator">
   <properties>
      <property name="output-style"
         value="in-source-comments" />
   </properties>
   <content-type>xml</content-type>
   <content-type>wsdl</content-type>
</artifact-comparator>
```

In this example, an artifact-comparator called "XMLComparator" is defined that utilizes the SampleXMLArtifactComparator class. A property specific to the SampleXMLArtifactComparator class "output-style" is specified. This artifact-comparator will be invoked for artifacts of with file extensions of "xml" and "wsdl".

Asset Validators

Asset Validators are classes that contain custom Asset validation logic to be executed synchronously during the Asset edit process. An Asset Validator class may perform extensive validation or updates on an Asset's metadata or artifact content beyond the scope of standard Asset Template-based metadata validation and overrides. Asset Validator instances may

optionally be qualified with an asset-filter, allowing the Asset Validator instance to be scoped to a specific subset of assets, and by a validation-context that determines when the validator will be applied within an asset's edit cycle. The absence of an asset-filter means that the Asset Validator instance is applicable to all Assets, while the absence of a validation-context means that the Asset Validator will run only during asset update and submission.

Function

When invoked, an Asset Validator instance is provided with an Event representing the context of the invocation, the Asset itself, and a collection of Validation Messages. The Asset Validator implementation may add Validation Messages of varying severities to the collection. These messages are targeted either to particular metadata elements or are designated as general Asset scoped messages. The Asset Validator instance may also update the Asset metadata[20].

Application

Depending on validation-context, when a user creates an asset or clicks "validate" on an asset during edit, Asset Validator instances that are applicable for the asset are selected and executed in the order they are defined in the LPC document. Each subsequent Asset Validator instance is provided with the resulting Asset and Validation Messages from the prior Asset Validator, allowing the validator implementations to augment or override the results of the previous Asset Validators[21].

Validation Context

The validation-context for an Asset Validator determines when in an asset's edit cycle the Asset Validator is to be invoked. There are six choices:

- ASSET_CREATION[22]
  Run at the initial creation of an asset when the asset is not created as a new version of another asset.
- ASSET_NEW_VERSION_CREATION[22]
  Run when an asset is created as a new version of another asset.
- ASSET_LIKE_CREATION[22]
  Run when an asset is created from another asset using the "Create Like…" option.
- ASSET_UPDATE (default)
  Run when a user either clicks "Validate" or submits the asset.
- ASSET_PARTIAL_UPDATE
  Run when a user saves an edit section (identifiers, classifiers, artifacts, or related assets) on the web UI[23]. Note that since it is not always required for a user to enter edit sections

[20] A SampleValidator class is provided in the extensions package showing the fundamentals of Asset Validator implementation. Additional information on concepts such as ValidationMessages may be found in the JavaDoc of relevant classes.

[21] Note that asset-filters of of subsequent validators will be applied to the modified version of the asset from previous validators.

[22] Since an Asset Validator defined with this validation-context is invoked just once during the creation of an asset, it should refrain from producing any severe error Validation Messages, as no attempt will be made to confirm that the user has corrected these errors prior to submitting the asset for publication, nor will such errors prohibit asset submission.

[23] This concept is supported on RM/PortM client UI's, but only when the entire asset is saved.

when creating an asset, this validation-context should normally be combined with ASSET_UPDATE to ensure that the Asset Validator is run at least once prior to submission.

- ASSET_TEMPLATE_TRANSITION
  Run immediately after a template is applied via submission post-processing (via the SetAssetTemplate listener).

Note that more than one validation-context may be set for an Asset Validator instance.

Validation Warnings Property

Since it may be desirable to alter asset-submission governance flow based on the existence of validation warnings, a pre-defined property is set on the catalog asset on completion of validation. This property has a key of "SOA:VALIDATION_WARNINGS_EXIST" and values of either "true" or "false". The property may be used in property-filters in the submission process configuration to alter flow based on the presence of validation warnings.

Defining an Asset Validator

Asset Validators are defined within the "asset-validators" element in the global section of the LPC document. Like Listeners, Asset Validators are configured with properties that the underlying Asset Validator instance may use in performing validation. An "asset-validator" element contains a unique name, an implementation class name, and properties specific to that class. Additionally, an asset-validator element may specify an asset-filter to identify the types of Assets that the AssetValidator instance is to be applied to, as well as a validation-context. Here is an example definition:

```
<asset-validator name="SampleValidator"
            class="com.soa.repositorymgr.extensions.SampleValidator">
      <properties>
        <property name="classifier-name" value="keyword" />
        <property name="classifier-value" value="new value" />
      </properties>
      <validation-context>ASSET_UPDATE</validation-context>
      <asset-filter-name>Services</asset-filter-name>
</asset-validator>
```

In this example, an Asset Validator called "SampleValidator" is defined that utilizes the extensions.SampleValidator class. Properties specific to the SampleValidator class are specified. The validation-context is set to ASSET_UPDATE, indicating that the validator is to be run during normal asset edit and submit. An Asset Filter name is specified, indicating that this Asset Validator instance will only be applied to assets matching the "Services" asset-filter.

Request Validators

Request Validators are similar in concept and implementation to Asset Validators. They validate and/or update Asset Requests and may be invoked when a request is submitted, approved or rejected. As with asset validators, request validator instances may optionally be qualified with an asset-filter, allowing the request validator instance to be scoped to requests for a specific subset of assets, and by a validation-context that determines when the validator will be applied. The absence of an asset-filter means that the request validator instance is applicable to requests

for all Assets, while the absence of a validation-context means that the request validator will run on request submission or update (including approval and rejection actions).

Function

When invoked, an Asset Validator instance is provided with a Context object representing the context of the invocation, the AssetRequest itself, and a collection of Validation Messages. The Asset Validator implementation may add Validation Messages of varying severities to the collection. Currently only messages designated as general request scoped messages are supported by the LifecycleManager clients. The Asset Validator instance may also update the request.

Application

Depending on validation-context, when a user clicks "submit", "save", "approve" or "reject" on a request, request validator instances that are applicable for the request are selected and executed in the order they are defined in the LPC document. Each subsequent request validator instance is provided with the resulting AssetRequest and validation Messages from the prior request validator, allowing the validator implementations to augment or override the results of the previous request validators.

Validation Context

The validation-context for a request validator determines when user actions on a request will cause the request validator to be invoked. There are two choices:

- REQUEST_SUBMITTER_UPDATE
  Run when the submitting user clicks "submit" on the request.
- REQUEST_APPROVER_UPDATE
  Run when an approver clicks "save", "approve" or "reject" on a request.

Note that more than one validation-context may be set for a request validator instance.

Defining a Request Validator

Request Validators are defined within the "request-validators" element in the global section of the LPC document. Like Listeners, request validators are configured with properties that the underlying request validator instance may use in performing validation. A "request-validator" element contains a unique name, an implementation class name, and properties specific to that class. Additionally, a request-validator element may specify an asset-filter to identify the types of Assets associated with the requests that the request validator instance is to be applied to, as well as a validation-context.
Here is an example definition:

```
  <request-validators>
      <request-validator name="ServiceRequestValidator"
class="ScriptedRequestValidator">
      <properties>
        <property name="script-id" value="scripts/ValidateServiceRequest.bsh" />
      </properties>
      <validation-context>REQUEST_SUBMITTER_UPDATE</validation-context>
      <asset-filter-name>Filter Services</asset-filter-name>
    </request-validator>
```

In this example, an request validator called “ServiceRequestValidator” is defined that uses the provided ScriptedRequestValidator class. Properties specific to the ScriptedRequestValidator class (in this case the script id) are specified. The validation-context is set to REQUEST_SUBMITTER_UPDATE, indicating that the validator is to be run when the request is submitted. An Asset Filter name is specified, indicating that this request validator instance will only be applied to requests for assets matching the “Services” asset-filter.

Value Sources

Value Source classes are responsible for producing a list of valid values for a property or asset classifier. This is a useful concept when the choice of values is dynamic or is coming from another system of record. Value Sources are defined in the LPC document and referenced either from property-definition elements (also in the LPC) or from define-classifier elements in the Asset Capture Template. The provider of a Value Source class may restrict its applicability to Property values, Classifier values, or support both. Additionally, the Value Source provider must choose whether the Value Source is to be accessed dynamically each time a referencing Classifier or Property is created, or whether its values will be cached by the Lifecycle Manager application until the next restart or upload of the LPC.

Defining a Value Source

Value Sources are defined within the “value-sources” element in the global section of the LPC document. Like Listeners, Value Source instances are configured with properties that the underlying Value Source instance may use in producing a list of values. A “value-source” element contains a unique name, an implementation class name, and properties specific to that class.
Here is an example definition:

```
<value-source name="APQC-OWLValueSource"
         class="OWLValueSource">
   <properties>
      <property name="owl-document-id" value="APQC-OWL.xml" />
      <property name="owl-root-class" value="PCF" />
   </properties>
</value-source>
```

In this example, a Value Source called “APQC-OWLValueSource” is defined that utilizes the OWLValueSource class. Properties specific to the APQC-OWLValueSource class are specified.

Additionally, the LPC author may choose the caching strategy for non-interactive value-sources. The caching strategy is defined in the <cache-strategy> child element. Three caching choices are available:

- **Dynamic**
  This strategy indicates that values from the value-source will be retrieved on demand with each call to the value-source. Values will not be cached. While this strategy assures the most recent value data (and may be the only practical choice for some value-sources), it may significantly impact UI performance if value retrieval is not efficient. Here is an example of a value-source with a dynamic cache strategy:

```
<value-source name="existing-values" class="ExistingClassifierValues">
  <cache-strategy>
```

```
      <dynamic/>
    </cache-strategy>
  </value-source>
```

- **Hourly**
  The hourly strategy indicates that values for the values-source will be cached and refreshed on an hourly basis. The LPC author may optionally specify the number of hours between refreshes using the "hours" attribute and a reference time from which to base the refresh intervals. Consider this example of an hourly cache-strategy:
  ```
  <cache-strategy>
          <hourly hours = "12" reference-time = "01:00:00"/>
  </cache-strategy>
  ```
  This strategy will result in values being cached and refreshed every 12 hours (the default value for hours is "1"). Since a reference-time of 1AM was provided, values will be refreshed at 1PM and 1AM every day. If no reference-time is specified, refresh times are based from the time of LPC upload.
- **Daily**
  The daily strategy indicates that values for the values-source will be cached and refreshed on a daily basis. The LPC author may optionally specify the number of days between refreshes using the "days" attribute and a refresh time indicating at time of day the refresh should occur. Consider this example of an daily cache-strategy:
  ```
  <cache-strategy>
          <daily days = "7" refresh-time = "01:00:00"/>
  </cache-strategy>
  ```
  This strategy will result in values being cached and refreshed every seven days (the default value for days is "1")[24]. Since a refresh-time of 1AM was provided, values will be refreshed at 1AM every seventh day. If no refresh-time is specified, it is defaulted to "00:00:00" (midnight).

If no cache-strategy is provided, caching behavior will fall-back to the strategy designated by the values-source class provided[25]. This will either be <dynamic> or the equivalent of <daily days = "0">, indicating that the cached values are only refreshed on LPC upload.

Referencing a Value Source from a Property Definition

Property-definition elements in the LPC reference Value Sources using a "value-source" attribute as shown in this example:

```
<property-definition name="RequestClassification" type="STRING"
        value-source="ClassificationOntology" />
```

In this example "ClassificationOntology" references a <value-source> element with matching "name" attribute. Enumerated values shown to the user for this property will come from the specified Value Source.

Referencing a Value Source from a Classifier Definitions

Value Sources may be referenced from either simple or Compound Classifier definitions:
Simple Classifiers

[24] Note that specifying "0" for the "days" attribute indicates that the cached values for the value-source will only be updated on LPC upload.
[25] Note that this matches the pre-6.4 release legacy behavior for value-source caching behavior.

Simple classifiers may reference Value Source by name using the "value-source" attribute:

```
<define-classifier name="business-classification" type="string" value-
source="ClassificationOntology"></define-classifier>
```

Compound Classifiers

As with simple classifiers, compound classifiers need only add a "value-source" attribute to reference a Value Source. Field definition is unchanged although it is assumed that the author of the classifier definition is aware of the field structure that will be used by the values returned from the Value Source and specifies the <field> elements accordingly. Here is an example compound classifier definition referencing a Value Source:

```
<define-compound-classifier name="apqc-domain" display-name="APQC Domain"
open="false" max-occurs="1" value-ordering="GDT" value-source="APQC-
OWLValueSource" help-text="The APQC domain to which this business process
applies">
    <fields>
      <field name="category" help-text="APQC category" />
      <field name="process group" help-text="APQC process group" />
      <field name="process" help-text="APQC process" />
      <field name="activity" help-text="APQC activity" />
    </fields>
</define-compound-classifier>
```

Importers

Importers are classes that allow an interactive import of Lifecycle Manager assets from a specified location or system or record such as a runtime registry. Importer classes have two responsibilities: providing a selection mechanism for potential-assets, and creating Lifecycle Manager assets from those selected entities. Importers defined in the LPC appear in the "Import Assets" page in Lifecycle Manager.

Defining an Importer

Importers are defined within the "importers" element in the global section of the LPC document. Like Listeners, importer instances are configured with properties that the instance may use in accessing asset data from an underlying system or record. An "importer" element contains a unique name, an implementation class name, and properties specific to that class. Here is an example definition:

```
<importer name="WSDL Importer" class="WSDLImporter">
 <properties>
   <property name="description" value="Import service from WSDL" />
   <property name="wsdl-artifact-containment" value="by-value" />
   <property name="service-asset-version" value="1.0" />
   <property name="service-asset-type" value="Web service" />
   <property name="service-asset-template" value="webservice" />
   <property name="wsdl-artifact-category" value="wsdl" />
  </properties>
</importer>
```

In this example, an Importer named "WSDL Importer" is defined that utilizes the WSDLImporter class. Properties specific to the WSDLImporter class are provided. Importer classes provided by Lifecycle Manager are documented in Appendix J.

Exporters

Exporters are classes that allow an interactive export of Lifecycle Manager assets from the results of an asset search. Exporter classes have the responsibility of converting assets to an external form and/or exporting the asset data to an external system of record. Exporters defined in the LPC appear are accessible to users through the "Export Assets" link on the search results page[26].

Defining an Exporter

Exporters are defined within the "exporters" element in the global section of the LPC document. Like importers, exporter instances are configured with properties that the instance may use in exporting asset data. An "exporter" element contains a unique name, an implementation class name, an attribute indicating the type of information returned and properties specific to that class. Here is an example definition:

```
<exporter name="Delimited Exporter"
          class="DelimitedFileAssetExporter"
          returns-file="true">
  <properties>
    <property name="full-header" value="true" />
    <property name="delimiter" value="#" />
  </properties>
</exporter>
```

In this example, an Exporter named "Delimited Exporter" is defined that utilizes the DelimitedFileAssetExporter class. Properties specific to the DelimitedFileAssetExporter class are provided. Exporter classes provided by Lifecycle Manager are documented in Appendix T. The "returns-file" attribute is used to indicate that the exporter class will either return a file or a series of AssetProcessingResult messages. The attribute is defaulted to "true" indicating that the application should expect a file to be returned.

Federated Systems

Federated Systems are used to define a remote system of record that Lifecycle Manager or Portfolio Manager libraries will communicate with. A Federated System not only contains connection information for a remote system but serves as an identity or alias for that system which can be referenced by Listeners, Importers, Artifact Sources, and Value Sources. This allows the actual connection information of the remote system to be changed without affecting referencing LPC elements or external reference properties held in assets. Since the name of a federated-system is its identifier, it must be unique and should not be changed.

Defining a Federated System

Federated Systems are defined within the "federated-systems" element in the global section of the LPC document. Like Listeners, Federated System instances are configured with properties that the instance may use in accessing the remote system. A "federated-system" element contains a unique name, an implementation class name, and properties specific to that class. Here is an example definition:

```
<federated-system
```

[26] In the 6.3.1 release only one Exporter may be defined per library.

```
  name="Repository Manager"
  class="FederatedRepository">
    <properties>
   <property name="remote-host"
      value="http://remotehost.com/PM" />
   <property name="remote-library-name"
      value="PortMRMTarget" />
   <property name="user" value="support" />
   <property name="password" value="thePassword" encrypt="true" />
    </properties>
</federated-system>
```

In this example, an Importer named "Repository Manager" is defined that utilizes the FederatedRepository class. Properties specific to the FederatedRepository class are provided. FederatedRepository classes provided by Lifecycle Manager and Portfolio Manager are documented in Appendix N.

Pending Request Filtering

Users possessing the currently required approver role for an Asset Request will see the request indicated in the "Pending Requests" count in the left navigation pane of the web client and also within the "Pending Requests" page. It is possible to apply additional filtering to the Asset Requests that appear in a user's pending request list through the use of Request Filters. Currently Request Filters are defined statically in the LPC Document and are applied globally to all approvers. Request Filters allow for filtering of requests based on request properties, associated asset classifiers, or a combination of the two.

Defining Request Filters

Request Filters are defined within the <request-filters> element in the global section of the Library Configuration Document. A <request-filter> element may specify a list of names of property-filters and asset-filters defined in their respective sections of the Library Configuration Document. For a pending Asset Request to comply with a particular Request Filter it must comply with at least one referenced Property Filter (if any exist) AND at least one Asset Filter (if any exist). The set of defined Request Filters are then logically OR'ed together to form the default request filtering behavior for all users.

Referenced property-filters and asset-filters may use context parameters (as described in the Context Replacement Parameter section of this document) to specify context specific filtering criteria. Note that when using context parameters in conjunction with request filtering, the user-context is the user for whom the requests are being filtered.

The following example provides filtering of Asset Requests based on the "database-architect" property matching the current user's account name.
Here is the request-filter declaration:

```
<request-filters>
   <request-filter name="My DatabaseArchitect Requests" >
     <property-filter-name>DatabaseArchitect Requests</property-filter-name>
   </request-filter>
</request-filters>
```

And here is the referenced property-filter. Note the use of the context parameter in the criteria; in this case the request's "database-architect" property must match the account name of the user for whom the requests are being filtered.

```
<property-filter name="DatabaseArchitect Requests">
   <property-criteria property-key="database-architect"
                      target="ASSET_REQUEST">
      <property-values>
         <value>{user.account}</value>
      </property-values>
   </property-criteria>
</property-filter>
```

Bypassing Submission Governance

With the 6.2 release of Lifecycle Manager it is possible to bypass Asset submission governance in cases where there are no "governed" changes to the catalog Asset. By default, all metadata elements of the Asset are considered "governed". However, by setting an element's "governed" attribute to "false" in the definition or capture template it's possible to designate the element as not subject to governance. When an Asset that contains no changes to governed elements is submitted, any configured submission governance will be bypassed and the asset will be published to the Library directly. Note that the submission of an asset with no changes meets this criterion and thus will bypass governance. It is possible to disable this feature for a Library by setting the Library property "ENABLE_GOVERNANCE_BYPASS" to "false". This may be accomplished using the SetLibraryProperty administration command.

WSDL resolution rules

Certain listeners (PolicyManagerPublisher) and validators (SchemaValidator, WSDLValidator, etc.) build up a WSDL or schema from the WSDL and schema artifacts and assets in a library as well as resolving them externally on the Internet. Certain validation messages and publication messages can confuse users if they do not understand these concepts. The following bullets describe how a WSDL's dependent WSDLs and schemas are resolved.

If the asset contains a packed/zipped artifact:

- All relative references are resolved relative to the including file in the zip. No other assets are involved in the resolution of missing or unresolvable includes and imports.

If the asset contains a normal WSDL message message definition artifact:

1. Attempt to resolve the import by namespace. Other assets in the library with a namespace classifier (configurable by listener/validator) that matches the imported namespace will be used. If only one such asset matches, its message definition or schema artifact will be used.
2. If there is no import namespace or there are multiple assets with the same namespace, the the import/include is resolved by location (see *Import/Include by Asset* step below)
3. If the import can't be found by an asset's location, resolve by location on web (see the *Import/Include by URL* step below)

4. The final step is to use the target namespace as a URL reference and try to resolve it on the web. If this step fails then the resource goes unresolved and a listener or validation error may occur.

Import/Include by Asset:

1. If the import/include location is fully qualified, search for assets with the same location
2. If the import/include location isn't fully qualified and the including resource has a fully qualified location, generate a fully qualified location based on the importing resource and relative import/include location and search for assets containing that location
3. If the import/include isn't fully qualified and the including resource isn't fully qualified (or absent), search for assets containing the import's relative location

Import/Include by URL:

1. If the import/include is fully qualified, resolve the location on the web
2. If the import/include isn't fully qualified and the including resource has a fully qualified location, generate a fully qualified location based on the importing resource and the relative imported location and use the web to resolve the location

Notes:
Importing resource means either an asset contains the WSDL/schema that is importing a schema or an external WSDL/schema is importing a schema.

Maintaining the LPC Document

Obtaining the Schema

The XML schema for the LPC document may be retrieved from a Lifecycle Manager installation by issuing the command *GetProcessConfigurationSchema* from the Lifecycle Manager Administrative Console. This command takes no parameters and ignores the selected library. It returns the schema in the output window. This schema may be used in schema-aware XML editing tools to build a LPC document.

Obtaining the Default Library Configuration

The default LPC document for a Lifecycle Manager installation may be retrieved by issuing the command *GetDefaultProcessConfiguration* from the Lifecycle Manager Administrative Console. This command takes no parameters and ignores the selected library. The default LPC document defines processes as follows:

- ASSET_DELETION
  This process integration point is disabled and the *ASSET_DELETION_APROVED* event mapped directly to the AssetDeletion listener.
- ASSET_SUBMISSION
  This process integration point is disabled and the *ASSET_SUBMISSION_APROVED* event mapped directly to the AssetSubmission listener.
- ASSET_ACQUISITION
  This process integration point is enabled. The default process flow is designed to preserve compatibility with acquisition behavior of previous Lifecycle Manager releases.

- ASSET_REVOCATION
  This process integration point is enabled. The default process flow is designed to preserve compatibility with revocation behavior of previous Lifecycle Manager releases.

It is recommended that either the default LPC document or a simpler skeleton LPC document available from the Lifecycle Manager support site be the starting point for LPC customization rather than starting from an empty document, so as to preserve default process behavior that is not being customized.

Obtaining the Current Library Configuration

The current (active) LPC document for a library may be retrieved by issuing the command *GetCurrentProcessConfiguration* from the Lifecycle Manager Administrative Console. This command takes no parameters. However, the library from which to retrieve the current LPC document must be specified, as each library can have its own configuration.

Setting the Current Library Configuration

The current (active) LPC document for a library may be updated by issuing the command *SetProcessConfiguration* from the Lifecycle Manager Administrative Console. This command requires that the "File Parameter" be set to a file containing the new LPC document. The selected library will determine which library to update. The command takes no other parameters. If there are errors in the updated LPC document they will be displayed in the command output window and the update will be aborted.

> **CAUTION:** Updating the LPC document is an action that should be performed with caution and a thorough understanding of the concepts in this document and of the effects the updated LPC document will have on the Lifecycle Manager library.
>
> Note that it is possible to break existing processes by using an incorrectly configured LPC document. Caution should also be taken not to reconfigure a process flow so as to leave existing Asset Requests "stranded" in a state that is no longer defined in the new process flow. For this reason, it is recommended that all existing Asset Requests for a particular process flow be allowed to complete (become inactive) prior to that process flow being updated.

Resetting the Default Library Configuration

The command *ResetProcessConfiguration* will reset the current (active) LPC document to the default LPC document. As with the *SetProcessConfiguration* command, caution should be used in invoking this command so as not to break in-process Asset Requests.

Appendix A: Internal Listeners

The following Listener classes are currently available in the Lifecycle Manager internal Listener Library:

AcquireRelatedAsset

- **Behavior:**
  Acquires assets that are the target of a relationship from the current asset, where the relationship matches the relationship names specified for this listener. Acquisitions of the related assets may optionally trigger an approval process. A recursion property can be set to follow relationships more than one level.
- **Usage Context:**
  This Listener should be used in the context of an asset acquisition when the ASSET_REGISTERED Event is raised, after the acquisition is complete.
- **Properties:**
  - *relationship_1*
    Relationship name to follow for acquisition, required. Note: multiple relationship_N properties may be specified. They must be consecutive numbers starting at 1. (Mandatory)
  - *create-request*
    If "true", an acquisition request is created for the acquisitions performed by this listener. If "false", acquisitions happen automatically. Default is "false". (Optional)
  - *acquisition-request-note*
    Note to add to the acquisition request. Allows parameter replacement. (Optional)
  - *recurse*
    If "true", multiple levels of relationships will be traversed in acquiring related assets. Default is "false". (Optional)
- **Prerequisites:**
  The acquisition of the current asset must be complete before this listener is invoked.
- **Return Codes:**
  - 0 – success
  - 1 – relationship is not defined
  - -1 – system error

AssetAcquisitionListener

- **Behavior:**
  Performs registration of an Asset to a Project.
- **Usage Context:**
  This Listener is generally triggered when *ASSET_ACQUISITION_REQUESTED, ASSET_ACQUISITION_APPROVED* or related Events are raised. If an acquisition approval process is enabled, this Listener should normally be triggered by the final approval of the acquisition request.
- **Properties:**
  None
- **Prerequisites:**
  None
- **Return Codes:**
  - 0 - success

AssetDeletionListener

- **Behavior:**
  Performs deletion of the specified Asset from the catalog and library.
- **Usage Context:**
  The default Library configuration triggers this Listener when the *ASSET_DELETION_REQUESTED* Event is raised. If a deletion approval process is enabled, this Listener should be triggered by the final approval of the deletion request.
- **Properties:**
  None
- **Prerequisites:**
  None
- **Return Codes:**
  - 0 - success

AssetRevocationListener

- **Behavior:**
  Removes registration of an Asset from a Project.
- **Usage Context:**
  This Listener is generally triggered when *ASSET_REVOCATION_REQUESTED, ASSET_REVOCATION_APPROVED* or related Events are raised. If a revocation approval process is enabled, this Listener should normally be triggered by the final approval of the revocation request.
- **Properties:**
  None
- **Prerequisites:**
  None
- **Return Codes:**
  - 0 - success

AssetSubmissionListener

- **Behavior:**
  Performs submission for publishing of the specified Asset from the catalog.
- **Usage Context:**
  The default Library configuration triggers this Listener when the *ASSET_SUBMISSION_REQUESTED* Event is raised. If a submission approval process is enabled, this Listener should be triggered by the final approval of the submission request.
- **Properties:**
  None.
- **Prerequisites:**
  The asset must be complete and valid for publishing.
- **Return Codes:**
  - 0 – success

AttachPropertiesToAsset

- **Behavior:**
  Attaches asset submission request properties to an asset as either classifiers or artifacts.
- **Usage Context:**
  As part of asset submission governance, those involved with the request may be required to populate the request with values. These values can then be populated onto the asset.
- **Properties:**
  - *locking-user*
    The user to use for updating the asset. By default the original submitter will be used. (Optional)
  - *document-id*
    The document source id of the configuration file defining which properties will be populated onto the asset. See the System Administrator Guide for more information on storing files into the document source. (Required)
- **Post-conditions:**
  The asset must be complete and valid for publishing after the asset's metadata is updated. Modification of asset metadata that affects governance flow may occur with this listener. For example, if a classifier is modified with a different value, and the workflow uses that value to determine which phase/flow to follow, you may see unexpected behavior where a post processing task you expect to run isn't run.
- **Document configuration:**
  The document the document-id refers to must be an XML file with a root "attach-properties-to-asset" element and child "property" elements. Each element defines a property to take from the request and attach to the asset. Each property name can only be listed once can be attached as an artifact and/or a classifier, as long as it is applicable.
- **property attributes:**
  - *name*
    The name of the asset request property (not display name) that will be attached to the asset.
  - *classifier-name*
    The name (not display name) of a classifier. If this attribute exists, the property will be populated as a classifier with a value corresponding to the property's value. File type properties cannot be attached to an asset.
  - *artifact-category*
    The artifact category of an artifact. If this attribute exists, the property will be populated as an artifact with a value corresponding to the property type. If the property is a file type, it will either be attached as a by-value artifact or by-reference artifact depending on whether the property contains a file or url. If the property is of a non-file type, it will be attached as a by-description artifact.
  - *replace*
    A Boolean indicating whether any existing artifacts or classifiers (of type *artifact-category* or *classifier-name*) should be removed before the property is attached. This should be used when either a classifier or artifact has a max-occurs="1" or you want to delete all previous values before attaching the new one.

Document example

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<attach-properties-to-asset>
    <property name="sign-off-property"
            classifier-name="sign-off-note"
            replace="false" />
    <property name="approval-document-property"
            artifact-category="approval-document"
            replace="true" />
</attach-properties-to-asset>
```

- **Return Codes:**
  - 0 – success
  - -1 – failure, or not all properties could be attached to the asset

BuildSchemaRelationships

- **Behavior:**
  This listener class is responsible for building relationships from Assets with XML Schema type artifacts to Assets representing referenced schemas. Relationships are generated from "import" and "include" references within an XML schema artifact. The listener attempts to find a target asset by searching for Assets with a matching "schemaLocation" or "targetNamespace" classifier value. If found a relationship is added to the source asset. If an Asset is not found and the referenced document is retrievable and is a schema document, the listener may optionally create a schema type Asset. In any other case, an unresolved relationship may be added (based on configuration properties) to the source Asset. The schemaLocation is first normalized as a path name based on the schemaLocation of the source document to provide a unique name for lookup. For example, if the schema location for the current document is /schemas/customer/cust.xsd and the imported schema specifies "../common/address.xsd", the schemaLocation that will be searched for a matching asset would be /schemas/common/address.xsd. Note that this requires the schemas that are referenced to exist in the catalog or the library prior to the including document getting processed.
- **Usage Context:**
  This listener is generally targeted at XML schema Assets in the submission process to force population of relationships to referenced schema Assets.
- **Properties:**
  - *locking-user*
    User Id used in modifying and creating Assets. If not specified, Lifecycle Manager Application User will be used. Allows parameter replacement. (Optional)
  - *target-artifact-category*
    Category of the schema artifact to generate relationships from. (Optional)
  - *adjust-schema-asset-name*
    If false, generated schema asset names will be the schema target namespace URI. If true and the target namespace is a valid URL, the path part of the URL will be used as the asset name with the "/" characters replaced with spaces. For example, if the target namespace URI is "http://schemas.xmlsoap.org/wsdl/soap/" and *adjust-schema-asset-name* is "true", the schema asset's name will be "wsdl soap". The default value for this property is "false". (Optional)

- *create-unresolved-relationships*
  Determines whether an unresolved relationship will be added to the target asset in the case where a referenced schema could not be resolved. The default value is "true". (Optional)
- *imported-schema-relationship*
  Actual Relationship name for the "imports" relationship. Default value is "imports". (Optional)[27]
- *included-schema-relationship*
  Actual Relationship name for the "includes" relationship. Default value is "includes". (Optional)
- *create-schema-assets*
  Indicates whether Assets should be automatically created for referenced schemas. Default value is "true". (Optional)
- *submit-schema-assets*
  Indicates whether created schema Assets should be submitted. Default value is "true". (Optional)
- *schema-asset-type*
  Asset type for created schema Assets. Default value is "XML Schema". (Optional)
- *schema-asset-template*
  Indicates the template to be assigned to created schema Assets. Default value is "XML Schema". (Optional).
- *schema-asset-version*
  Version to use for generated schema assets. Default value is "1.0". (Optional)
- *schema-asset-owning-group*
  Owning Group for created schema assets. If not provided, owning group defaults to the reporting Group of the *locking-user*. (Optional)
- *schema-asset-description*
  Description to be used for created schema Assets. Defaults to "Auto-generated XML Schema Asset". (Optional)
- *schema-asset-overview*
  Overview to be used for created schema Assets. Defaults to "Auto-generated XML Schema Asset". (Optional)
- *schema-artifact-category*
  Category to be used for the schema document artifact in created schema Assets. Defaults to "schema". (Optional)
- *schema-artifact-containment*
  Type of containment for the schema document artifact in created schema Assets. Allowed values are "by-value" and "by-reference". Defaults to "by-reference". (Optional)
- *schema-document-version*
  Value used for the schema version classifier. Defaults to "1.0" (Optional).

[27] Note that if the asset's current template does not support the current relationship type, the relationship will be added as "user-defined". This applies to all relationships added by this listener.

- *schema-version-classifier*
  Classifier name for the schema version classifier. Defaults to "schemaVersion" (Optional).
- *schema-namespace-classifier*
  Classifier name for the schema target namespace classifier. Defaults to "targetNamespace". Note that this will also be used in searching for existing related schema Assets. (Optional).
- *schema-location-classifier*
  Classifier name for the schema location classifier. Defaults to "schemaLocation". (Optional)
- *schema-submit-note*
  Note used for submission of schema Assets. Defaults to "Asset submitted by BuildSchemaRelationships Listener". (Optional)
- *process-schema-includes*
  If true, XML schema includes will be processed as well as schema imports. Default is false. (Optional)

- **Prerequisites:**
  The *target-artifact-category* artifact should reference a valid XML schema document.
- **Return Codes:**
  - 0 – success
  - 2 – schema artifact not found
  - 3 – artifact is not an XML Schema
  - 4 – too many assets found for a given import/include

BuildWSDLRelationships

- **Behavior:**
  This listener class is responsible for building relationships from Assets with WSDL or BPEL type artifacts to Assets representing referenced schemas or services. Relationships may be generated for the following:
  - Referenced Schemas
    Relationships are generated to Assets representing schemas referenced as namespaces from within the target document. WSDL "imports" elements are leveraged when present to locate the referenced schema document. Assets for referenced schemas may optionally be created by this listener.
  - Referenced Services
    Relationships are generated to Assets representing Web services or abstract service declarations referenced as namespaces from within the target document. WSDL "imports" elements are leveraged when present to locate the referenced service document. Service documents are identified by the presence of a "definitions" element.
    Assets for referenced services may optionally be created by this listener.
  - Deployed Services
    Deployed Services are a special case of Referenced Services as described above. If a referenced service is an *abstract* service (meaning there are no bindings declared) *and* the target Asset of the listener represents a service deployment (having bindings declared), a special "deploys" relationship will be generated.

This relationship may have special meaning to other processes such as UDDI publishing.

Note that relationships and associated Assets will only be created in cases where the referenced document is successfully retrieved (or an Asset already existed for the document) *and* the document is either a schema or WSDL style document (has a "definitions" root element). Assets are matched based on the schemaLocation classifier. The specified schemaLocation or location attribute of the referenced document is first is first normalized as a path name based on the location of the source document to provide a unique name for lookup. For example, if the schema location for the current document is /services/customer/cust.wsdl and the imported schema specifies "cust.xsd", the schemaLocation that will be searched for a matching asset would be /services/customer/cust.xsd. Note that this requires the schemas/WSDLs that are referenced to exist in the catalog or the library prior to the including document getting processed.

- **Usage Context:**
  This listener is generally targeted at Web service Assets in the submission process to force population of relationships to referenced schema and service Assets.
  Note this listener can only be run in a disconnected environment (where referenced documents are not accessible) if both create-schema-assets and create-service-assets properties are set to false.
- **Properties:**
  - *locking-user*
    User Id used in modifying and creating Assets. If not specified, Lifecycle Manager Application User will be used. Allows parameter replacement. (Optional)
  - *target-artifact-category*
    Category of the schema artifact to generate relationships from. (Optional)
  - *adjust-schema-asset-name*
    If false, generated schema asset names will be the schema target namespace URI. If true and the target namespace is a valid URL, the path part of the URL will be used as the asset name with the "/" characters replaced with spaces. For example, if the target namespace URI is "http://schemas.xmlsoap.org/wsdl/soap/" and *adjust-schema-asset-name* is "true", the schema asset's name will be "wsdl soap". The default value for this property is "false". (Optional)
  - *adjust-service-asset-name*
    This property has the same semantics as *adjust-schema-asset-name.* The default value for this property is "false". (Optional)
  - *referenced-schema-relationship*
    Relationship name for a referenced schema. Default value is "related-schema". (Optional)[28]
  - *included-schema-relationship*
    Relationship name for a schema include relationship. Default value is "related-schema". (Optional)

[28] Note that if the asset's current template does not support the current relationship type, the relationship will be added as "user-defined". This applies to all relationships added by this listener.

- *referenced-service-relationship*
  Relationship name for a referenced service. Default value is "related-service". (Optional)
- *service-deployment-relationship*
  Relationship name to be used in the special case of a Web service Deployment Asset's relationship to a referenced Web service asset. Defaults to "deploys". (Optional)
- *service-location-classifier*
  Classifier name to be used to look up a Web service asset by the location specified in a WSDL import statement. Default is null, which will not look up by location, only namespace. (Optional)
- *create-schema-assets*
  Indicates whether Assets should be automatically created for referenced schemas. Default value is "true". (Optional)
- *create-service-assets*
  Indicates whether Assets should be automatically created for referenced services. Default value is "true". (Optional)
- *submit-schema-assets*
  Indicates whether created schema Assets should be submitted. Default value is "true". (Optional)
- *submit-service-assets*
  Indicates whether created service Assets should be submitted. Default value is "true". (Optional)
- *schema-asset-type*
  Asset type for schema Assets. Default value is "XML Schema". (Optional)
- *service-asset-type*
  Asset type for service Assets. Default value is "Web service". (Optional)
- *schema-asset-template*
  Indicates the template to be assigned to created schema Assets. Default value is "XML Schema". (Optional).
- *service-asset-template*
  Indicates the template to be assigned to created service Assets. Default value is "WebService". (Optional).
- *schema-asset-version*
  Version to use for generated schema assets. Default value is "1.0". (Optional)
- *service-asset-version*
  Version to use for generated service assets. Default value is "1.0". (Optional)
- *schema-asset-owning-group*
  Owning Group for created schema assets. If not provided, owning group defaults to the reporting Group of the *locking-user*. (Optional)
- *service-asset-owning-group*
  Owning Group for created service assets. If not provided, owning group defaults to the reporting Group of the *locking-user*. (Optional)
- *schema-asset-description*
  Description to be used for created schema Assets. Defaults to "Auto-generated XML Schema Asset". (Optional)

- *service-asset-description*
  Description to be used for created service Assets. Defaults to "Auto-generated XML Web Service Asset". (Optional)
- *schema-asset-overview*
  Overview to be used for created schema Assets. Defaults to "Auto-generated XML Schema Asset". (Optional)
- *service-asset-overview*
  Overview to be used for created service Assets. Defaults to "Auto-generated XML Web Service Asset". (Optional)
- *schema-artifact-category*
  Category to be used for the schema document artifact in created schema Assets. Defaults to "schema". (Optional)
- *wsdl-artifact-category*
  Category to be used for the WSDL document artifact in created service Assets and for interrogating a referenced service to determine if the asset is an abstract service. Defaults to "wsdl". (Optional)
- *schema-artifact-containment*
  Type of containment for the schema document artifact in created schema Assets. Allowed values are "by-value" and "by-reference". Defaults to "by-reference". (Optional)
- *wsdl-artifact-containment*
  Type of containment for the WSDL document artifact in created service Assets. Allowed values are "by-value" and "by-reference". Defaults to "by-reference". (Optional)
- *schema-document-version*
  Value used for the schema version classifier. Defaults to "1.0" (Optional).
- *schema-version-classifier*
  Classifier name for the schema version classifier. Defaults to "schemaVersion" (Optional).
- *schema-namespace-classifier*
  Classifier name for the schema target namespace classifier. Defaults to "targetNamespace". Note that this will also be used in searching for existing related schema Assets. (Optional).
- *service-namespace-classifier*
  Classifier name for the service target namespace classifier. Defaults to "targetNamespace". Note that this will also be used in searching for existing related service Assets. (Optional).
- *schema-location-classifier*
  Classifier name for the schema location classifier. Defaults to "schemaLocation". (Optional)
- *schema-submit-note*
  Note used for submission of schema Assets. Defaults to "Asset submitted by BuildWSDLRelationships Listener". (Optional)
- *service-submit-note*
  Note used for submission of service Assets. Defaults to "Asset submitted by BuildWSDLRelationships Listener". (Optional)

- *process-schema-imports*
  If true, XML schema imports will be processed as well as WSDL imports. Default is false. (Optional)
- *process-schema-includes*
  If true, XML schema includes will be processed as well as WSDL imports. Default is false. (Optional)

- **Prerequisites:**
  The *target-artifact-category* artifact should reference a valid WSDL or BPEL document.
- **Return Codes:**
  - 0 – success
  - 2 – artifact not found
  - 3 – artifact is not WSDL
  - 4 – too many assets found for a given import/include

CreateAssetSurveyListener

- **Behavior:**
  This listener will create a "survey" asset and associate it with the asset in the event via a related asset. The managers of the project associated with this event will be sent a notification email pointing them to the new asset.
- **Usage Context:**
  Used when creating a survey for an asset.
- **Properties:**
  - *locking-user*
    User account name used to create the survey asset. If not specified, Lifecycle Manager Application User will be used. Allows parameter replacement. (Optional)
  - *asset-survey-asset-type*
    Asset type of the asset survey asset. (Required)
  - *asset-survey-template-name*
    "Pending" capture template for the survey asset. Default is "Asset_Survey_Template". (Optional)
  - *asset-survey-relationship-name*
    Relationship to create between the survey asset and the event asset. Default is "survey". (Optional)
  - *asset-survey-mail-template-id*
    Mail template id for the notification email. (Required)
    The mail template is passed the following parms:
    - {0} = event type
    - {1} = project name
    - {2} = asset name
    - {3} = asset version
    - {4} = asset detail page URL
    - {5} = survey asset name
    - {6} = survey asset version
    - {7} = survey asset detail page URL
  - *asset-survey-last-notified-classifier-name*

Classifier name of the date last notified. Must be a date type. (Optional)
    - *asset-survey-project-classifier-name*
      Name of the classifier that will be set to the project name. (Optional)
    - *asset-survey-status-classifier-name*
      Name of the classifier that will hold the status. (Optional)
    - *asset-survey-status-initial-value*
      Initial value of the status classifier. (Optional)
    - *asset-survey-overview-text*
      Text to use for the asset overview. (Required)
- **Prerequisites:**
  Please look at above properties for more information.
- **Return Codes:**
  - 0 – success
  - 1 – Configuration error.
  - 2 – System error.

DeleteAssetSurveys

- **Behavior:**
  This listener will delete survey assets associated with an asset. It determines which assets to delete by finding all assets that contain a relationship to the event asset with a given relationship name. It is designed to be used to remove survey assets for an asset that is being deleted. If an error occurs, this listener will not roll back deletes and may leave some survey assets remaining. In the case of federated libraries, this listener will need to be configured in all libraries that may have acquisitions of either local or remote assets.
- **Usage Context:**
  This Listener should be used in the context of an asset deletion. It should be triggered from the ASSET_DELETED event.
- **Properties:**
    - *asset-survey-asset-type*
      Asset type of the asset survey asset. (Required)
    - *asset-survey-relationship-name*
      Relationship name between the survey asset and the event asset. (Required)
    - *locking-user*
      User account name used to delete the survey asset. If not specified, Lifecycle Manager Application User will be used. Allows parameter replacement. (Optional)
    - *deletion-note*
      Note added to the audit trail for each survey asset that is deleted. Default is "Deleted automatically by DeleteAssetSurveys listener". (Optional)
- **Return Codes:**
  - 0 – success

GenericPolicyValidationListener

- **Behavior**:

- Walks through all the policy assets related to the service asset as specified in the listener properties and validates service asset artifact(s) specified with each policy. Policy validation will apply the associated Policy criteria to the specified service asset artifact by using the Validator class specified on the Policy Criteria. Validation results are added to the report along with the valid or invalid result message. If the asset doesn't pass the validation on all the "required policy criteria", the "Validation Succeeded" Boolean on the asset is updated if specified in the listener.

- **Properties**:
  - *validation-policy*
    Name of the relationship that specifies validation policies on service asset. Default value is *"*policy*"*. (Required)
  - *service-artifact*
    Artifact that needs to be validated. Default value is *"*wsdl*"*. (Required)
  - *validation-class*
    Name of the classifier that specifies Algorithm name in Policy Criteria asset. Defaults to *"*Validation Class*"*. (Optional)
  - *report-artifact*
    Validation report Artifact name. (Optional)
  - *validation-succeeded*
    Name of the classifier which will be set to true or false if all the policies were validated. (Optional)
  - required-criteria - - Name of the relationship that specify required validation policy criteria on Policy asset. (Optional)
  - *optional-criteria*
    Name of the relationship that specify optional validation policy criteria on Policy asset. (Optional)
  - *xpath-expression*
    Name of the classifier that specifies XPATH expression name in Policy Criteria asset. Defaults to "xpath-expression". (Optional)
  - *valid-message*
    Name of the classifier that specifies Result message in Policy Criteria asset if Validation Succeeded. (Optional)
  - *Invalid-message*
    Name of the classifier that specifies Result message in Policy Criteria asset if Validation Failed. (Optional)
  - *locking-user*
    Name of the user who will publish the service asset with changes. If not specified, Lifecycle Manager Application User will be used. Allows parameter replacement. (Optional)
- **Return Codes:**
  - 0 – success

GenericRequestHandler

- **Behavior**:
  May perform the following actions:

- Notify users of a specified Group role using a specified mail template.
- Notify the submitter of the request using a specified mail template.
- Update the current state of the Asset Request to a specified state
- Add a pending Approval Instance to the Asset Request for a specified Group role.
- Add an entry to the history of the Asset Request
- Lock or unlock the Asset associated with the triggering Event
- Inactivate the Asset Request.

- **Usage Context**:
  This Listener class is used in implementing standard approval processes (for designated process integration points) where each new state of the request requires the notification of a set of users of a particular role on the context Group. An initial instance of this listener is generally triggered by the *ASSET_DELETION_REQUESTED* or *ASSET_SUBMISSION_REQUESTED* Events. Subsequent instances are triggered by Events associated with approvals or rejections of the Asset Request[29].
- **Properties**:
  - *request-type*
    Indicates the type of Asset Request that should be modified. Note that there may be multiple Asset Requests of different types active for a single Asset. Currently defined request types are "ASSET_SUBMISSION" and "ASSET_DELETION". (Required)
  - *request-state*
    Indicates the new state that the Asset Request should be placed in when the Listener is triggered. The state should be descriptive of the Asset Request's current state in the approval process. For example: "Pending Architect Approvals". This property is optional. If omitted, the Asset Request will be left in its current state. Allows parameter replacement.
  - *recipient-role*
    Indicates the Group role of the Users that will be notified. The Asset Request will also be modified to indicate that it is pending approval by a user of the specified Group role[30].
    This property is optional, but must be provided if the *recipient-message-id* property is specified. If provided, the property must name an existing Group role (either a predefined Lifecycle Manager role such as "Project Manager" or a role declared in the "group-roles" element of the LPC document.
  - *owning-group-recipient*
    This property is a Boolean value indicating that the recipient-role specified applies to the owning Group of the Asset rather than the context Group of the triggering Event[31]. This property is optional and is specified as "true" to indicate

[29] Example usage of the GenericRequestHandler class can be found in Appendixes C and D.
[30] Note that the prefix "ACQ:" may be used with a group role to indicate that role players of the specified role in projects that have acquired the asset are to be notified. For example: "ACQ:Project Manager". Events triggered by role players in an acquiring project will similarly be preceded with the "ACQ:" prefix. For example: "ASSET_SUBMISSION_ACQ:Project Manager_APPROVED". This behavior allows role players of an acquiring project to participate in the submission process for the update of an acquired asset.
[31] Since the context Group and asset owning Group for the ASSET_DELETION and ASSET_SUBMISSION processes are always the same, it is not be necessary to specify this property.

that the Asset's owning Group should be used for determining recipients. If not specified, the property defaults to "false"[32].

- *terminate-request*
  This property is a Boolean value indicating that the request should be terminated when the Listener is invoked. It is commonly specified on rejection or final approval of an Asset Request. This property is optional and is specified as "true" to indicate that the Asset Request should be terminated.
- *recipient-message-id*
  This property indicates the mail template to be used to notify recipients by email. Lifecycle Manager currently provides a generic mail template for approver notification with an id of "APPROVER_ACTION_REQUIRED". This property is optional. If not specified, recipients will not be notified. See the "Commands" section in the Lifecycle Manager System Administration guide for more information about retrieving mail template information.
- *submitter-message-id*
  This property indicates the mail template to be used to notify the submitter of the Asset Request by email. Lifecycle Manager currently provides two generic mail templates for submitter notification with ids of "SUBMITTER_REQUEST_APPROVED" and "SUBMITTER_REQUEST_REJECTED". This property is optional. If not specified, the submitter will not be notified.
- *lock-asset*
  This property is a Boolean property indicating whether the Asset should be locked or unlocked when this Listener is invoked. A value of "true" indicates the Asset should be locked. In this case the *locking-user* property may also be specified. A value of "false" indicates that the Asset should be unlocked regardless of which User currently has the Asset locked. If this property is not specified, the Asset will be left in its current lock state.
- *locking-user*
  This property may be specified if the *lock-asset* property is set to "true". The value of the property should be set to an existing user ID in the library being updated. If not specified and *lock-asset* is true, it will default to the Lifecycle Manager Application User. Allows parameter replacement. (Optional)
- *history-entry*
  This property is a note to be added to the Asset Request history. This property is optional. Allows parameter replacement.
- *history-user*
  This property indicates the user to associate with the history entry if history-entry is specified. If not specified and history-entry is specified, the Lifecycle Manager Application User will be used. Allows parameter replacement. Allows parameter replacement. (Optional)
- *property-constraint-set*
  This property specifies the default property-constraint-set to be applied to the target Asset Request. See Appendix I for additional details

[32] As of the Logidex 5.5 release, if recipient-role is specified as "Asset Owner" and this property is not specified it will default to "true".

- *approval-property-constraint-set*
  This property specifies the property-constraint-set to be applied to the target Asset Request when a user chooses to approve the request. If not specified, the property-constraint-set specified in the *property-constraint-set* property is used for the approval scenario.
- *rejection-property-constraint-set*
  This property specifies the property-constraint-set to be applied to the target Asset Request when a user chooses to reject the request. If not specified, the property-constraint-set specified in the *property-constraint-set* property is used for the rejection scenario.
- *reject-request*
  This a Boolean property is used to indicate that the GenericRequestHandler should automatically reject the asset request (as opposed to this occurring interactively with a human approver). A value of "true" indicates the request should be rejected. (Optional)
- *rejecting-role*
  This property is only meaningful if *reject-request* is true. It is used to set the role that will be associated with the rejecting user. The role specified need not be a defined group role. This property is mandatory if *reject-request* is true.
- *rejecting-user*
  This property is only meaningful if *reject-request* is true. It is used to set the user associated with the rejection. If not specified, the rejecting user will default to *history-user*, then to the Lifecycle Manager Application User. (Optional)
- *approve-request*
  This a Boolean property is used to indicate that the GenericRequestHandler should automatically approve the asset request (as opposed to this occurring interactively with a human approver). A value of "true" indicates the request should be approved. (Optional)
- *approving-role*
  This property is only meaningful if *approve-request* is true. It is used to set the role that will be associated with the approving user. The role specified need not be a defined group role. This property is mandatory if *approve-request* is true.
- *approving-user*
  This property is only meaningful if *approve-request* is true. It is used to set the user associated with the approval. If not specified, the approving user will default to *history-user*, then to the system user. (Optional)

- **Return Codes:**
  - 0 – success
  - 1 – indicates that the *recipient-message-id* property was specified but no users of the specified *recipient-role* could be found to be notified.
  - 2 – indicates that an "ACQ:" prefixed role was used but that there were no Projects that had acquired the asset.

HandleError

- **Behavior:**
  Generates an email message notifying users with the Library Administrator role (in the

context Library of the triggering event), inactivating an asset request if present, optionally unlocking the asset, optionally setting an Asset Request history note, and creating a zip file containing the log entries at the time of the error. The zip file creation can be avoided by setting the three line-count properties to zero.

- **Usage Context:**
  Commonly used to notify Library Administrators of an unrecoverable system error during LPC processing.
- **Properties:**
  - message-id
    This property indicates the mail template to be used to generate the email to the Library Administrators. (Required)
    The parameters passed into this mail template will be as follows:
    - {0} = event type
    - {1} = acting user name
    - {2} = acting user email
    - {3} = asset name (if the triggering event pertains to an asset)
    - {4} = asset version (if the triggering event pertains to an asset)
    - {5} = asset Id (if the triggering event pertains to an asset)
    - {6} = Group name (if the triggering event has a context Group)
    - {7} = recipient's name
    - {8} = recipient's role (will always be "Library Administrator")
    - {9} = request URL
    - {10} = zip file name
  - *request-state*
    Indicates the state that the Asset Request will be placed in when the request is terminated. It is an arbitrary string that will be displayed for the state of the terminated request. This property is optional. If this property is not specified, the Asset Request state will be set to "Error". Allows parameter replacement.
  - *unlock-asset*
    This property is a Boolean property indicating whether the Asset should be unlocked when this Listener is invoked. A value of "true" indicates the Asset should be unlocked. A value of "false" indicates that the Asset should not be unlocked. If this property is not specified, the Asset will be left in the state it was in when this listener was invoked.
  - *history-entry*
    This property is a note to be added to the Asset Request history. For example: "Request automatically terminated due to system error". This property is optional. If this property is not specified, a history entry will not be added. Allows parameter replacement.
  - *application-line-count*
    This property indicates the number of lines to archive into a zip file from the application log at the time of the error. Default is 10000. (Optional)
  - *incident-line-count*
    This property indicates the number of lines to archive into a zip file from the incident log at the time of the error. Default is 10000. (Optional)

- *policymanager-line-count*
  This property indicates the number of lines to archive into a zip file from the integrated PolicyManager server log at the time of the error. Default is 10000. (Optional)

- **Prerequisites:**
  The mail template specified with the *message-id* property must exist.
- **Return Codes:**
  - 0 – always returns zero

JMSListener

Note that this listener requires a JMS TopicConnectionFactory and Topic to be pre-configured in the application server that the SOA application is running on. Contact SOA support for additional details on configuring JMS communication.

- **Behavior:**
  Generates a JMS Message from the triggering event to a specified JMS Topic. Specifically, a JMS MapMessage is sent with the fields and properties of the event as properties in the message. When applicable, the following properties corresponding to event fields are added as properties to the MapMessage:
  - SOA_ACTION – the event name
  - SOA_ASSET_ID – associated asset Id
  - SOA_ACTIVE_ASSET_ID – consuming asset id for acquisition events
  - SOA_ACTIVE_PROJECT_ID – active project id
  - SOA_REQUEST_ID – associated request id
  - SOA_COMPONENT – event component
  - SOA_LIBRARY_ID – library Id
  - SOA_OWNING_GROUP_ID – Group Id for asset production events

  Event-specific properties are set using the event property name prefixed with "SOA_PROPERTY_".
- **Usage Context:**
  Used to propagate selected SOA events as messages to a JMS Topic. Selection of SOA events is accomplished through the use of an <event-filter> on the <action> that enables this listener. Multiple listener instances may be configured to send different event messages to different JMS topics.
- **Properties:**
  - *factory-name*
    JNDI name of the JMS TopicConnectionFactory (Optional – defaults to "jms/TopicConnectionFactory")
  - *topic-name*
    JNDI name of the JMS Topic to publish messages to (Mandatory)
  - *persistent-delivery*
    Indicates whether messages should be published persistently. Values are "true" and "false" (Optional – defaults to "true").
  - *priority*
    Indicates the priority used when publishing JMS messages. Values are "0" through "9" (Optional – defaults to JMS default message priority).

- *validate-connection*
  Indicates whether the JMS connection should be tested on LPC upload. Values are "true" and "false" (Optional – defaults to "true").

- **Prerequisites:**
  None
- **Return Codes:**
  - 0 – success
  - -1 – failure

Message

Note: This listener has been superseded by the SendMessage listener, which uses the Message framework introduced in the 6.2 release and eliminating the need to define the message subject and body as properties.

- **Behavior:**
  Sends the specified email to the specified recipients. Allows context based parameter replacement within recipients, subject and body of email.
- **Usage Context:**
  Commonly used to provide custom event-driven email notifications.
- **Properties:**
  - *to-recipients*
    Users/roleplayers that will be included on the "to" line of the generated email. This property is mandatory. Recipients are comma separated and are describe in Appendix K.
  - *cc-recipients*
    Users/roleplayers that will be included on the "CC" line of the generated email. See comment on parameter options below. This property is optional. Recipients are comma separated and are describe in Appendix K.
  - *bcc-recipients*
    Users/roleplayers that will be included on the "BCC" line of the generated email. See comment on parameter options below. This property is optional. Recipients are comma separated and are describe in Appendix K.
  - *subject*
    Subject line of the email. See comment on parameter options below. This property is mandatory. Replacement parameters are described in Appendix K.
  - *body*
    Body of the email. See comment on parameter options below. This property is mandatory. Replacement parameters are described in Appendix K.
- **Prerequisites:**
  None
- **Return Codes:**
  - 0 – success

NotifyActingUser

- **Behavior:**
  Generates an email message to the user that caused the event triggering this listener.

- **Usage Context:**
  Commonly used to notify users of the results of some action that they performed.
- **Properties:**
  - *message-id*
    This property indicates the mail template to be used to generate the email to the acting user . (Required)
    The parameters passed into this mail template will be as follows:
    - {0} = event type
    - {1} = acting user name
    - {2} = asset name (if the triggering event pertains to an asset)
    - {3} = asset version (if the triggering event pertains to an asset)
    - {4} = asset Id (if the triggering event pertains to an asset)
    - {5} = Group name (if the triggering event has a context Group)
- **Prerequisites:**
  The mail template specified with the *message-id* property must exist.
- **Return Codes:**
  - 0 – success

NotifyDesignatedParty

- **Behavior:**
  Generates an email message to a specified email address.
- **Usage Context:**
  Used to notify a specific party or email account of the results of some action that occurred in Lifecycle Manager.
- **Properties:**
  - *message-id*
    This property indicates the mail template to be used to generate the email to the specified email account. (Required)
    The parameters passed into this mail template will be as follows:
    - {0} = event type
    - {1} = acting user name
    - {2} = acting user email
    - {3} = asset name (if the triggering event pertains to an asset)
    - {4} = asset version (if the triggering event pertains to an asset)
    - {5} = asset Id (if the triggering event pertains to an asset)
    - {6} = Group name (if the triggering event has a context Group)
  - *email*
    Email account that message should be sent to.
- **Prerequisites:**
  The mail template specified with the *message-id* property must exist. The specified email address must be of valid format.
- **Return Codes:**
  - 0 – success

NotifyGroupRolePlayers

- **Behavior:**
  Generates an email message to the users that play a specified role on the context Group of the triggering event.
- **Usage Context:**
  Commonly used to notify role players of some action that was performed on their associated Group or sub-Group.
- **Properties:**
  - *message-id*
    This property indicates the mail template to be used to generate the email to the acting user . (Required)
    The parameters passed into this mail template will be as follows:
    - {0} = event type
    - {1} = acting user name
    - {2} = acting user email
    - {3} = asset name (if the triggering event pertains to an asset)
    - {4} = asset version (if the triggering event pertains to an asset)
    - {5} = asset Id (if the triggering event pertains to an asset)
    - {6} = Group name (if the triggering event has a context Group)
    - {7} = recipient's name
    - {8} = recipient's role
  - *recipient-role*
    Group role name of roleplayers that are to be recipients of this notification.
  - *owning-group-recipient*
    This property is a Boolean value indicating that the recipient-role specified applies to the owning Group of the Asset rather than the context Group of the triggering Event. This property is optional and is specified as "true" to indicate that the Asset's owning Group should be used for determining recipients. If not specified, the property defaults to "false" [33].
- **Prerequisites:**
  The mail template specified with the *message-id* property must exist.
- **Return Codes:**
  - 0 – success
  - 1 – indicates that no recipients of the specified recipient-role could be found to be notified.

NotifyLibraryAdministrators

- **Behavior:**
  Generates an email message notifying users with the Library Administrator role (in the context Library) of the triggering event.
- **Usage Context:**
  Commonly used to notify Library Administrators of some error or situation that has occurred that requires their intervention.

[33] As of the Logidex 5.5 release, if recipient-role is specified as "Asset Owner" and this property is not specified it will default to "true".

- **Properties:**
  - *message-id*
    This property indicates the mail template to be used to generate the email to the Library Administrators. (Required)
    Note that the parameters passed into this mail template will be as follows:
    - {0} = event type
    - {1} = acting user name
    - {2} = acting user email
    - {3} = asset name (if the triggering event pertains to an asset)
    - {4} = asset version (if the triggering event pertains to an asset)
    - {5} = asset Id (if the triggering event pertains to an asset)
    - {6} = Group name (if the triggering event has a context Group)
    - {7} = recipient's name
    - {8} = recipient's role (will always be "Library Administrator")
- **Prerequisites:**
  The mail template specified with the *message-id* property must exist.
- **Return Codes:**
  - 0 – success

NotifyUsageControllers

- **Behavior:**
  Generates an email message notifying users with the Usage Controller role (in the context Library) of the triggering event.
- **Usage Context:**
  Commonly used to notify Usage Controllers of some error or situation that has occurred that requires their intervention.
- **Properties:**
  - *message-id*
    This property indicates the mail template to be used to generate the email to the Usage Controllers. (Required)
    The parameters passed into this mail template will be as follows:
    - {0} = event type
    - {1} = acting user name
    - {2} = acting user email
    - {3} = asset name (if the triggering event pertains to an asset)
    - {4} = asset version (if the triggering event pertains to an asset)
    - {5} = asset Id (if the triggering event pertains to an asset)
    - {6} = Group name (if the triggering event has a context Group)
    - {7} = recipient's name
    - {8} = recipient's role (will always be"Usage Controller")
- **Prerequisites:**
  The mail template specified with the *message-id* property must exist.
- **Return Codes:**
  - 0 – success

PublishArtifactToClearCase listener

- **Behavior:**
  The PublishArtifactToClearCase listener creates and checks in artifacts to a Base ClearCase configuration specified by a ClearCaseSystem utilizing a dynamic view. Snapshot views are not supported by this listener. The listener is complimented by the ClearCaseArtifactSource which allows all Lifecycle Manager clients (browser, rich client interfaces, and extension implementations) to access files stored in ClearCase. The resulting by-reference artifact reference will have the following format:
  `soa://<cc>/pvob:<pvob>/fpath:<fpath>`
  The listener can be inserted into the governance flow by drag and dropping the publish to clearcase task from the palette of Configuration Designer onto the governance phase of a process. If the task is not offered on the palette, it needs to be added to your tasktemplates.xml file of your configuration project. Take the following steps to add the task:
  1. Create a copy of your current tasktemplates.xml file into your project.
  2. Select the tasktemplates.xml file, right click to raise the context menu and choose Refresh (use the SOA provided Refresh and not the generic F5 Refresh). This will download the default tasktemplates.xml file shipped with the library.
  3. You will be prompted for a library. Choose the appropriate library.
  4. Open the refreshed tasktemplates.xml and search for a task-template of "Publish to ClearCase". Copy the task-template element and paste it into the copy of the tasktemplates.xml you created in step 1.
  5. Adjust the federated-system-name property value as needed (see comments in the tasktemplate) in your copy of the tasktemplates.xml you created in step 1.
  6. Delete the tasktemplates.xml file.
  7. Rename your copy of the tasktemplates.xml you created in step 1 to "tasktemplates.xml".
  8. Close and re-open your lpc.lpc file. The "Publish to ClearCase" task should be available.
- **Usage Context:**
  Useful in pushing artifacts into a Base ClearCase system as part of asset governance.
- **System Requirements:**
  The Lifecycle Manager application will function as a ClearCase client and thus requires the Rational ClearCase Client. The Lifecycle Manager application needs typical ClearCase user rights to allow such things as creation of a view. The Lifecycle Manager application must have a supporting ClearCaseSystem configured and referenced by the federated-system-name property.
- **Properties:**
  - *federated-system-name*
    (Required) The name of the federated system that listener will populate the artifact into
  - *source-artifact*
    (Required) The artifact-category identifying the artifact which is to be placed into the federated system. It is required that the artifact is by-value.

- *target-artifact*
  (Required) The artifact-category of the ClearCase controlled artifact created by the listener.
- *target-fpath-classifier*
  (Required) The compound classifier which identifies the ClearCase fpath for the artifact
- *target-fpath-prefix*
  An optional fpath prefix what will be concatenated with the value of the target-fpath-classifier where path segments are separated by one of the following delimiters: "\", "/", "|". Ex: "soa_software\plugins\".
- *delete-source-artifact*
  If true (default), the listener will delete the source-artifact from the asset upon creation of the target-artifact
- *locking-user*
  User Id used in modifying and creating Assets. If not specified, Lifecycle Manager Application User will be used. Allows parameter replacement. (Optional).

- **Return Codes:**
  - 0 – success
  - *-1 – error*
- **Example Artifact reference:**
  - *soa://cc/pvob:proddev_pvob/baseline:prod_project_01_24_2008/component:helloworld/fpath:helloworld.wsdl*

PublishArtifactToWebDAV listener

- **Behavior:**
  The PublishArtifactToWebDAV listener creates and checks in artifacts to a WebDAV compliant system. It can be configured to check-in artifacts in a variety of ways – see the properties section for supported methods. This listener will create a directory structure in WebDAV in the following format *webdavpath/assetname/assetversion/artifactname* or *webdavpath/assetname/assetversion/artifactcategory/artifactname* depending on the use-artifact-directory property. The listener can also be configured to create a URL to the new WebDAV reference (replacing the original artifact).
- **Usage Context:**
  Useful in pushing artifacts into a WebDAV system as part of asset governance or for a more fine grained artifact based version control mechanism.
- **System Requirements:**
  The Lifecycle Manager application will function as a WebDAV client. The WebDAV server must support a small subset of WebDAV options, including but not limited to GET/PUT/PROPFIND/PROPPATCH.
- **Properties:**
  - *federated-system-name*
    (Required) The name of the federated system that listener will populate the artifact into. This federated system should be a WebDAVSystem.
  - *webdav-path*

(Required) The directory where the asset directories will be created. This directory should be absolute and should start with a preceeding slash. Example: "/svn/trunk/Assets"

- *target-artifact-category*
  The artifact-category of the artifact that will be pushed into WebDAV. This property can be omitted to use all artifacts.
- *replace-target-artifact*
  If true, after the listener has published the artifact in WebDAV, it will remove the artifact and replace it with a link to the file in WebDAV.
  Default: false
- *publish-by-reference-artifacts*
  If true, this will retrieve the contents of any by-reference artifacts (matching the target-artifact-category) and store them in WebDAV.
  Default: false
- *use-artifact-directory*
  If true, the listener will use a directory based on the artifact's display name. If a display name doesn't exist, it will use the artifact's category name. The format used will be *webdavpath/assetname/assetversion/artifactname* if false or *webdavpath/assetname/assetversion/artifactcategory/artifactname* if true.
  Default: false
- *convert-to-lowercase*
  If true, the listener will create all content in WebDAV with lowercase names. This includes the asset name/version, artifact directory, and artifact name. Caution using this as certain file names should have their case preserved (Java source files). In certain instances, you may end up with files in WebDAV that have differ only by case, and this can be problematic on filesystems that make no such distinction (NTFS).
  Default: false
- *locking-user*
  User Id used in modifying and creating Assets. If not specified, Lifecycle Manager Application User will be used. Allows parameter replacement. (Optional).

- **Return Codes:**
  - 0 – success
  - *-1 – error*

RegExValidator

- **Behavior:**
  Validates asset version and/or asset classifiers using Perl 5 regular expressions. See http://search.cpan.org/dist/perl/pod/perlre.pod for a description of expression syntax. If errors are found, an email is sent to the acting user describing the errors.
- **Usage Context:**
  Designed to be used on asset submission (Event *ASSET_SUBMISSION_REQUESTED*).
- **Properties:**

- *message-id*
  This property indicates the mail template to be used to generate the email to the acting user . (Required)
  The parameters passed into this mail template will be as follows:
  - {0} = event type
  - {1} = library name
  - {2} = asset name
  - {3} = asset version
  - {4} = asset id
  - {5} = group name
  - {6} = error message
- *metadata_1*
  Indicates the type of metadata to be validated. Value is of the form:
  asset.version
  or
  classifier.*classifier-name*
  Note: many may exist by changing _1 to _2, etc. Numbers must be consecutive. At least one must be specified.
- *expression_1*
  A valid Perl 5 regular expression. One to one correspondence with metadata_N property.
- *message_1*
  A description of the validation check being made which is sent as part of the email when an error occurs in validation. Corresponds to metadata_N property. Default is "<*metadata_N*> does not match expression <*expression_N*>". (Optional)

- **Prerequisites:**
  None.
- **Return Codes:**
  - 0 – success
  - 1 – validation error
  - 2 – metadata item not found in the asset and no validation error occurred

RemoteAssetPublisher

Behavior:
Publishes assets into a remote library. This target library may be either a Lifecycle Manager or Portfolio Manager library. The RemoteAssetPublisher listener leverages metadata mappings defined in the asset definition template to allow transformation of asset metadata to the format of the target library.

Usage Context:
Commonly used in flowing asset data between libraries with differing asset types and formats. For example, this listener is used to facilitate integration between SOA software's Portfolio Manager and Lifecycle Manager products.

- **Properties:**
  - *federated-system-name*
    Name of a federated-system of class FederatedRepository defined in the

federated-systems section of the LPC document that represents the remote target library. (Mandatory)

- *Remote-federated-system-name*
  Name of the federated-system of class FederatedRepository in the remote library that will be used in call-backs to the source asset. This remote FederatedRepository should be connected to this (the source) library. This property is optional but is required to support non-copied by-value artifacts.
- *submit-assets*
  Submit assets in the remote library. Defaults to false (Optional)
- *create-note*
  Asset creation note used in remote library. (Optional)
- *update-note*
  Asset update note used in remote library. (Optional)
- *submit-note*
  Submission note used in remote library is submit-assets is true (Optional)
- *name-pattern*
  A pattern using replacement parameters used in generating the name of the remote asset, if not provided the source asset name will be used.(Optional)
  These replacement parameters may be used in patterns:
  library_asset.urn
  library_asset.url
  library_asset.name
  library_asset.version
  library_asset.property.<property name>
  library_asset.classifier.<classifier name>
- *version-pattern*
  A pattern using replacement parameters used in generating the version of the remote asset, if not provided the source asset version will be used. See above for applicable replacement parameters (Optional).
- *description-pattern*
  A pattern using replacement parameters used in generating the description of the remote asset, if not provided the source asset description will be used. See above for applicable replacement parameters (Optional).
- *asset-overview*
  text for overview of remote asset, if not provided source asset overview will be used (Optional)
- *asset-template*
  Capture template to be used for asset in remote library
- *owning-group*
  Owning group for remote asset, if not provided the federated-system user's reporting group will be the owning group (Optional).
- *asset-type*
  Default asset-type classifier used for remote asset if a mapped asset-type value is not found. (Conditional: must be provided in the case where a mapped asset-type classifier is not found in the source asset).

- *force-updates*
  Force updated to remote asset even if locked by another user. Defaults to false. (Optional).
- *mapping-id*
  mapping-id to use in accessing metadata element mappings from Asset Definition Template. Default mapping-id is "RAP". (Optional).
- *owning-group-classifier*
  Name of the classifier in the source asset from which to look up the remote owning group id. Optional: If not specified, remote assets will be owned by the production-group of the user specified in the specified FederatedRepository.
- *remote-reference-category*
  Indicates the artifact category to be used to store a reference in the remote asset back to the source asset. Optional: If not set, the remote asset will not have a reference to the source asset.
- *remote-reference-name*
  Used when *remote-reference-category* is provided to further specify the name of the artifact used to store a reference in the remote asset back to the source asset. Optional: If not specified, *remote-reference-category* will be used as the artifact name. Note that this property is ignored if *remote-reference-category* is not specified.

- **Note on by-value artifacts**
  For target asset artifacts (as designated by mapping-id) marked as by-value in the remote asset template the default behavior is to create a custom artifact in the remote asset that references the source artifact in source library. This behavior requires that the remote-federated-system-name property is specified. If this property is not set, the by-value artifacts will not be propagated. Note that it is possible to specify copy behavior for a by-value artifact by appending ":copy" onto the category name in the artifact mapping. For example:
  <define-artifact category="documentation" display-name="Documentation" >
  <external-mapping key= "RAP" value = "documentation:copy"/>
  </define-artifact>
  Using this approach does not require the remote-federated-system-name property be set
- **Prerequisites:**
  A federated-system for the remote library must be defined.
- **Return Codes:**
  - 0 – success
  - 2 – Could not communicate with remote library
  - 3 - Validation errors were encountered in creating the remote asset.
  - 4 – Locking conflict while attempting to lock remote asset

RemoveAssetClassifier

- **Behavior:**
  Removes values for a classifier from an asset in the catalog.
- **Usage Context:**
  Used to update the value of a classifier on an asset. Useful for asset state transitions as the asset proceeds through its lifecycle.

- **Properties:**
  - *classifier-name*
    The name of the classifier to remove values from.
  - *classifier-value*
    The value or values to remove from the classifier. Multiple classifier values may be specified in this property by separating values with "::". This property is optional, if not specified all values for the specified classifier will be removed from the asset. Allows parameter replacement. (Optional)
  - *locking-user*
    The user account to use in performing the updates to the asset. The value of the property must be set to an existing user account in the library being updated. If not specified, Lifecycle Manager Application User will be used. Allows parameter replacement. (Optional)
  - *Note*
    Note text to be associated with the change to the asset. Allows parameter replacement. (Optional)
- **Prerequisites:**
  None.
- **Return Codes:**
  - 0 – success
  - 1 – Asset metadata validation errors occurred
  - 2 – classifier was not found on asset and no update was performed

RunCommand

- **Behavior:**
  Runs Lifecycle Manager Administrative Commands from within the Library configuration framework. Target Commands are limited to those that do not require a "File" parameter[34].
- **Usage Context:**
  Used to configure running of Commands in "batch" mode (see Appendix F) or as the result of some Lifecycle Manager Event.
- **Properties:**
  - *command-name*
    This property indicates the name of the Command to run.
    This property is mandatory.
  - *parm-<n>*
    Parameters to the specified command are designated with property names parm-1, parm-2, parm-3, etc. These properties correspond to the "Parameter <n>" fields that would normally be specified when the Command is run interactively through the "Lifecycle Manager Internal Maintenance Page".
- **Prerequisites:**
  The specified Command must not require a "File Parameter" since it is not possible to specify file parameters using this Listener.
- **Return Codes:**

[34] See the Lifecycle Manager System Administrators Guide for additional information on Commands.

- 0 – success

ScriptListener

- **Behavior:**
  Invokes a script, a user-definable set of instructions. Current scripting languages supported are BeanShell (a Java-based scripting language) and Jython (A Python-based scripting language for Java).
- **Usage Context:**
  The behavior of the script is up to the end-user to decide what to do. Potential scenarios include updating an asset or performing validation. A variable called "event" (type: com.logiclibrary.external.notification.Event) is passed into the script that would normally be available to a custom listener.
- **Properties:**
  - *script-id*
    This is the name of the script to invoke. This script must be uploaded using the StoreDocument command. See the System Administration guide for more information on the StoreDocument command.
  - *script-type*
    This determines which interpreter to execute the script. Valid values are "jython" or "beanshell". If this parameter is not specified, the interpreter used depends on the script-id's extension (.py or .bsh).
- **Prerequisites:**
  None.
- **Return Codes:**
  Set the variable "returnCode" in the script to make a different value available for processing by the workflow. If the returnCode variable isn't set, it will assume a value of 0. Do not "return" a value from the script as one would normally do from the function of a procedural programming language.
  - 0 – success

Scripting:
Information on the scripting languages can be found on their respective websites: BeanShell (http://www.beanshell.org/) or Jython (http://www.jython.org/). Integrating with SOA Software classes is possible, but they must be imported as needed. See the extensions.zip file (available from the SOA support site) for documentation on classes available for use within a script as well as sample scripts. Here is a short script that modifies the return code based on the Asset's name (available from the event).

BeanShell:

```
assetName = event.getProperty("ASSET_NAME");
returnCode = assetName.equalsIgnoreCase("Fail") ? 1 : 0;
```

Jython:

```
assetName = event.getProperty("ASSET_NAME")
if assetName.lower() == "fail":
    returnCode = 1
else:
    returnCode = 0
```

SendMessage

- **Behavior:**
  Sends the specified message to the specified recipients. Allows context based parameter replacement within recipients and message template.
- **Usage Context:**
  Commonly used to provide custom event-driven email notifications.
- **Properties:**
  - *context-recipients*
    Roleplayers that the message will be sent to. Context recipients are comma separated and are describe in Appendix K. Note that either this property and/or the email-recipients property must be specified.
  - *email-recipients*
    Specific emails that the message will be sent to. Email addresses are comma separated. Note that either this property and/or the context-recipients property must be specified.
  - *message-id*
    The id (name) of the message to be sent. Message templates are found in the Document Repository under the directory "messages" and may be maintained using Configuration Designer. For example, to use a message template file defined in the messages directory called "PROJECT_MANAGER_NOTIFICATION.html", the message-id property would be set to "PROJECT_MANAGER_NOTIFICATION". This property is mandatory.
  - *require-active-request*
    If set to "true" the message will be sent only if the request referenced in the event context is still active. This property is optional and defaults to "false".
- **Prerequisites:**
  None
- **Return Codes:**
  - 0 – success

SetAssetClassifier

- **Behavior:**
  Sets a classifier value or values on an asset in the catalog.
- **Usage Context:**
  Used to update the value of a classifier on an asset. Useful for asset state transitions as the asset proceeds through its lifecycle.
- **Properties:**
  - *classifier-name*
    The name of the classifier to set
  - *classifier-value*
    The value or values to set the classifier to. Multiple classifier values may be specified in this property by separating values with "::".Allows parameter replacement.

- *update-semantics*
  Defines which action should be taken in the case where the classifier already exists on the target asset. Possible values and their meaning are:
  - "add"
    Indicates that the values specified in the classifier-value property should be added to those already present.
  - "replace"
    Indicates that the values specified in the classifier-value property replace those already present.
  - "ignore"
    Indicates that the classifier values should not be updated if there are values already present.
- *locking-user*
  The user account to use in performing the updates to the asset. The value of the property must be set to an existing user account in the library being updated. If not specified, Lifecycle Manager Application User will be used. Allows parameter replacement. (Optional)
- *note*
  Note text to be associated with the change to the asset. Allows parameter replacement. (Optional)
- *submit-asset*
  Indicates that the asset should be submitted after the classifier is added. Defaults to false. (Optional)
- *submit-note*
  Used in the case when *submit-asset* is true as the asset submission (publish) note. Allows parameter replacement. (Optional)

- **Prerequisites:**
  The *classifier-name* specified must exist in the GDT.
- **Return Codes:**
  - 0 – success
  - 1 – Asset metadata validation errors occurred
  - 2 – "ignore" semantics were chosen and no update was performed

SetAssetRelationship

- **Behavior:**
  Sets an asset relationship on an asset in the catalog.
- **Usage Context:**
  Used to automatically set asset relationships. Useful for automatically assigning relationships to validation Policy assets.
- **Properties:**
  - *relationship-name*
    The name of the relationship to set.
  - *related-asset-name*
    The name of the related asset. This property is mandatory if related-asset-id is not provided. Allows parameter replacement.

- *related-asset-version*
  The version of the related asset. This property is mandatory if related-asset-id is not provided. Allows parameter replacement.
- *related-asset-id*
  The assetId of the related asset. This property is mandatory if related-asset-name and related-asset-version are not provided. Allows parameter replacement.
- *update-semantics*
  Defines which action should be taken in the case where the relationship already exists on the target asset. Possible values and their meaning are:
  - "add"
    Indicates that relationship defined by this listener should be added to those already present.
  - "replace"
    Indicates that the relationship defined by this listener replace those already present.
  - "ignore"
    Indicates that relationship defined by this listener should not be updated if there are relationships of the same name already present.
- *locking-user*
  The user account to use in performing the updates to the asset. The value of the property must be set to an existing user account in the library being updated. If not specified, Lifecycle Manager Application User will be used. Allows parameter replacement. (Optional)
- *Note*
  Note text to be associated with the change to the asset. Allows parameter replacement.

- **Prerequisites:**
  The *classifier-name* specified must exist in the GDT.
- **Return Codes:**
  - 0 – success
  - 1 – Asset metadata validation errors occurred
  - 2 – "ignore" semantics were chosen and no update was performed

SetAssetTemplate

- **Behavior:**
  Sets the capture template for an asset in the catalog.
- **Usage Context:**
  Used to automatically apply different templates to an asset as it progresses through its lifecycle.
- **Properties:**
  - *asset-template-name*
    The name of the template to apply. Allows parameter replacement.
  - *locking-user*
    The user account to use in performing the updates to the asset. The value of the property must be set to an existing user account in the library being updated. If

not specified, Lifecycle Manager Application User will be used. Allows parameter replacement. (Optional)
    - *note*
      Note text to be associated with the change to the asset. Allows parameter replacement. (Optional)
    - *apply-if-valid*
      A Boolean property indicating whether the specified template should be applied if it causes the asset's meta-data to become invalid. A value of "true" indicates that the template should not be applied if the asset becomes invalid. Default value is "false". (Optional)
    - *staged*
      A Boolean property indicating whether the specified template should be immediately applied or staged for application on the next edit after asset publish. Default value is "true". (Optional)

- **Prerequisites:**
  A capture template with *asset-template-name* must exist in the library.
- **Return Codes:**
    - 0 – success
    - 1 – Asset metadata validation errors occurred
    - 2 – template was not found

SubscribeSubmitter

- **Behavior:**
  Subscribes an Asset submitter to the published Asset.
- **Usage Context:**
  Generally configured to be triggered at Asset publish time by the *ASSET_AUTO_PUBLISH* and *ASSET_MANUAL_PUBLISH* Events.
- **Properties:**
  None.
- **Prerequisites:**
  None.
- **Return Codes:**
    - 0 - success

UpdateAssetSurveyInfoListener

- **Behavior:**
  This listener will take survey asset information and update information in the target asset.
- **Usage Context:**
  Used when a completed survey asset has been submitted.
- **Properties:**
    - *locking-user*
      User account name used to update the target asset. If not specified, Lifecycle Manager Application User will be used. Allows parameter replacement. (Optional).

- *asset-survey-relationship-name*
  The relationship between the survey asset and the target asset. Default is "survey". (Optional)
- *asset-update-failure-mail-template-id*
  Mail template id for the notification email if asset cannot be updated. (Required)
  The parameters passed into this mail template will be as follows:
  - {0} = event type
  - {1} = asset name
  - {2} = asset version
  - {3} = asset detail page URL
  - {4} = reason message
  - {5} = role name
- *asset-update-failure-notify-role*
  Role to send a failure notification email to. Default is "Project Manager". (Optional)

Peer rating properties - If any of these ratings are not required, then do not specify any of these properties.

- *target-asset-peer-rating-classifier-name*
  Classifier name of the peer rating in the target asset. (Optional)
- *target-asset-peer-rating-count-classifier-name*
  Classifier name of the peer rating count in the target asset. (Optional)
- *survey-asset-peer-rating-classifier-name*
  Classifier name of the peer rating in the survey asset. (Optional)

Survey Status Properties

- *asset-survey-status-classifier-name*
  Name of the classifier that will hold the status. (Required)
- *asset-survey-status-initial-value*
  Initial value of the status classifier. (Required)
- *asset-survey-status-complete-value*
  Value of the status classifier for complete state. (Required)

Forum topic properties

- *survey-asset-forum-artifact-category_1*
  Artifact category of an artifact to be posted to a forum. (Optional)
  Note: many may exist by changing _1 to _2, etc. Numbers must be consecutive.
- *target-asset-forum-topic_1*
  Forum topic of an asset forum on the target asset which will contain the survey forum artifact contents. One to one correspondence with survey-asset-forum-artifact-category_N property.

- **Prerequisites:**
  A capture template with *asset-template-name* must exist in the library.
- **Return Codes:**
  - 0 – success
  - 1 – Configuration error.

- 2 – System error.
- 3 – Not supported error.

WebLayersValidator

- **Behavior:**
  Validates specified artifacts for compliance with a WebLayers Governor. Optionally adds validation results as artifacts on the Asset in the case of validation errors.
- **Usage Context:**
  Used either on adding or updating of an artifact on an Asset (Events *ARTIFACT_CREATED* or *ARTIFACT_UPDATED*) or on asset submission (Event *ASSET_SUBMISSION_REQUESTED*).
- **Properties:**
  - *target-artifact-category*
    Category names of the artifact(s) to be validated. Either a single Category name or a comma-delimited list of names may be specified. This is a mandatory property.
  - *result-artifact-category*
    Category of the validation result artifacts. Validation result artifacts will be by-reference and created with the specified category and a name of the form "<result-artifact-name-prefix (see next property)><target artifact name> _WebLayers<Success/Failure><timestamp>". The by-reference artifact created, will be a link to the WebLayers UI that will contain more information on the validation.
    This is an optional property. If not specified, the validation results will not be stored as artifacts on the Asset.
  - *result-artifact-name-prefix*
    A string to append to the result artifact name, if it is created. This allows multiple WebLayersValidator listeners to qualify the assets they create using a user configurable value.
    This is an optional property. If not specified, there will be no prefix added to the resulting artifact name.
  - *replace-results*
    If specified with a value of "true" and a result-artifact-category has been specified, previous XML validation result artifacts will be removed from the specified result-artifact-category prior to validation. This facilitates a scenario where result artifacts are automatically removed from the asset as validation errors are corrected. The default is "false".
  - *consolidate-results*
    If specified with a value of "true", the validation results will only include one validation result artifact (if created) containing the combined status of all artifacts processed (success if every artifact passes, or failure otherwise). If specified with a value of "false" a result artifact will be created for every target artifact processed. The default is "false".
  - *exclude-successful-results*
    If specified with a value of "true" validation result artifacts will not be added for successfully validated artifacts. If this property is set to "false", all validation

messages will be added regardless of an artifact's validation success. The default is "false".

- *validation-succeeded*
  This property should be set to the name of a Boolean classifier. This classifier will be set according to the results of the validation. If the validation was successful, the classifier will be set to "true", otherwise if there was a problem, the classifier will be set to "false". This property may be omitted if no classifiers should be updated accordingly.
- *locking-user*
  This property is used to indicate the User to use if the Asset is not locked[35]. This property is optional, if not specified and validation results are to be added to the asset and the asset is not locked, the Lifecycle Manager Application User will be used to lock the asset. Allows parameter replacement.
- *metadata-value-x (where x is a sequence starting at 1)*
  The metadata-value-x and metdata-name-x properties are specified in pairs and can be specified any number of times. The purpose is to pass Lifecycle Manager asset information to WebLayers. WebLayers expects to receive metadata information a name/value pair format. The metadata-name property should be something recognizable by the WebLayers governor (e.g. "com.Lifecycle Manager.AssetVersion". The value can be either hardcoded or one of the following values, which are replaced with asset specific information when the listener runs.
  asset.name
  asset.version
  asset.description
  classifier.*classifiername*

  The following is an example of passing the asset name and version to WebLayers:
  <property name="metadata-name-1" value="Name" />
  <property name="metadata-value-1" value="asset.name" />
  <property name="metadata-name-2" value="Version" />
  <property name="metadata-value-2" value="asset.version" />
- *weblayers-server*
  The URL of the WebLayers server (e.g. *http://www.example.com/weblayers-center*). This property is required.
- *weblayers-use-ssl*
  If specified with a value of "true" SSL will be used to communicate with WebLayers. Note: specifying https:// in the server name will not automatically accomplish this.
- *weblayers-username*
  The user used to connect to the WebLayers server. This property is required.
- *weblayers-password*
  The password for the corresponding user used to connect to the WebLayers server. This property is required.

[35] If the Asset is locked, the result artifacts will be added by the locking User

- *weblayers-governance-id*
  This property indicates which WebLayers governor will be used to validate the artifacts in this asset. This property is required.
- *weblayers-max-retries*
- *weblayers-retry-wait-millis*
  These properties affect how Lifecycle Manager handles timeouts from the WebLayers backend. If WebLayers cannot connect for some reason it will retry up to *weblayers-max-retries* times. If it needs to retry for some reason it will wait weblayers-retry-wait-millis milliseconds before doing so. The default is to not retry.
- *weblayers-on-behalf-of*
  This is the user that will be used as an actor for the validation. This user may be different than the *weblayers-username* property. If this property is not specified, the user id submitting the artifact for validation will be used instead.
- *weblayers-use-proxy*
- *weblayers-proxy-host*
- *weblayers-proxy-port*
- *weblayers-proxy-password*
- *weblayers-proxy-realm*
- *weblayers-proxy-user*
  These properties affect if Lifecycle Manager needs a proxy to connect to WebLayers. *weblayers-use-proxy* can be set to "true" or "false" and affects whether the other proxy parameters are used. The other proxy parameters are standard proxy settings and not specific to WebLayers
- *weblayers-proxy-auth-type*
  This parameter affects which type of authentication is used to connect to the proxy. It may take one of the following values: "none", "basic", "digest", "ntlm".

- **Prerequisites:**
  None.
- **Return Codes:**
  - 0 - success
  - 1 – indicates that the validation found errors
  - 99 – indicates that no artifacts in the specified category were found

XMLArtifactValidator

- **Behavior:**
  Validates specified XML artifacts for correct XML syntax and compliance with specified schemas. Optionally adds validation results as artifacts on the Asset in the case of validation errors.
- **Usage Context:**
  Used either on adding or updating of an XML artifact on an Asset (Events *ARTIFACT_CREATED* or *ARTIFACT_UPDATED*) or on asset submission (Event *ASSET_SUBMISSION_REQUESTED*).
- **Properties:**

- *target-artifact-category*
  Category names of the artifact(s) to be validated. Either a single Category name or a comma-delimited list of names may be specified. This is a mandatory property.
- *result-artifact-category*
  Category of the validation result artifacts. Validation result artifacts will be created with the specified category and a name of the form
  "<target artifact name> _ValidationResults <timestamp>"
  This is an optional property. If not specified, the validation results will not be stored as artifacts on the Asset.
- *locking-user*
  This property is used to indicate the User to use if the Asset is not locked[36]. This property is optional, if not specified and validation results are to be added to the asset and the asset is not locked, the Lifecycle Manager Application User will be used to lock the asset. Allows parameter replacement. (Optional)
- *replace-results*
  If specified with a value of "true" and a result-artifact-category has been specified, previous XML validation result artifacts will be removed from the specified result-artifact-category prior to validation. This facilitates a scenario where result artifacts are automatically removed from the asset as validation errors are corrected.
- *<XML namespace>*
  An arbitrary number of namespace-to-schema mappings may be added using the XML namespace name as the property key and a URL to the associated schema as the associated property value. For example:

```
<property name="http://www.w3.org/2001/XMLSchema"
value="http://www.w3.org/2001/XMLSchema.xsd" />
```

  The default namespace (noNamespace) schema is specified using the key "noNamespaceSchema".
  Any namespace referenced in the XML artifacts to be validated that does not correspond to a schema mapping will result in a validation failure. Note that schema mappings for basic XML, SOAP, and WSDL documents are defined by default and do not need to be specified as properties. These default mappings are as follows:
  - Name: http://schemas.xmlsoap.org/wsdl/http/
    Value: http://schemas.xmlsoap.org/wsdl/http/
  - Name: http://schemas.xmlsoap.org/wsdl/soap/
    Value: http://schemas.xmlsoap.org/wsdl/soap/
  - Name: http://www.w3.org/2001/XMLSchema
    Value: http://www.w3.org/2001/XMLSchema.xsd
  - Name: http://schemas.xmlsoap.org/soap/encoding/
    Value: http://schemas.xmlsoap.org/soap/encoding/
  - Name: http://schemas.xmlsoap.org/wsdl/mime/
    Value: http://schemas.xmlsoap.org/wsdl/mime/
  - Name: http://schemas.xmlsoap.org/wsdl/
    Value: http://schemas.xmlsoap.org/wsdl/

[36] If the Asset is locked, the result artifacts will be added by the locking User

- **Prerequisites:**
  None.
- **Return Codes:**
  - 0 - success
  - 1 – indicates that the validation found errors
  - 99 – indicates that no artifacts in the specified category were found
- **Schema retrieval notes:**
  For proper validation, Lifecycle Manager needs access to schemas on the Internet. There may be issues fetching these schemas as many companies require access to the Internet through a proxy or have firewall rules in place that limit the ability of servers to connect to the Internet[37]. If this resembles your environment there are two options available: configuring to use a proxy, or configure Lifecycle Manager to use local schemas.
  - *Use a proxy*
    Using a proxy requires configuring the JVM of the application server with the appropriate settings. The JVM needs to have the following options set (modify as needed):

    ```
    -DproxySet=true -DproxyHost=proxyhost.example.com -DproxyPort=8080
    ```

    Two more JVM arguments will need to be set if proxy authentication is required:

    ```
    -DproxyUser=username -DproxyPassword=userpassword
    ```

  - *Use local schemas*
    Using local schemas requires copying all necessary xsd files from remote servers to a local server that Lifecycle Manager has access to. The list presented below represents the minimum schemas needed to validate a WSDL file, which are included with the Lifecycle Manager web application. You may need to change "localhost" to something else if your network configuration prevents such access. If your XML files introduce new namespaces you'll need to include a namespace property in the LPC document and make available all additional XSD files.

    The following list are the XML namespace elements that need to be added to the XMLArtifactValidator section:

    ```
    <property name="http://schemas.xmlsoap.org/wsdl/http/"
           value="http://localhost/Repository Manager/schema/wsdl_http.xml" />
    <property name="http://schemas.xmlsoap.org/wsdl/soap/"
           value="http://localhost/Repository Manager/schema/wsdl_soap.xml" />
    <property name="http://www.w3.org/2001/XMLSchema"
           value="http://localhost/Repository Manager/schema/XMLSchema.xsd" />
    <property name="http://schemas.xmlsoap.org/soap/encoding/"
           value="http://localhost/Repository Manager/schema/soap_encoding.xml" />
    <property name="http://schemas.xmlsoap.org/wsdl/mime/"
           value="http://localhost/Repository Manager/schema/wsdl_mime.xml" />
    <property name="http://schemas.xmlsoap.org/wsdl/"
           value="http://localhost/Repository Manager/schema/wsdl.xml" />
    <property name="http://www.w3.org/XML/1998/namespace"
           value="http://localhost/Repository Manager/schema/xml.xsd" />
    ```

[37] Note that the timeout used by Java in fetching data from a URL can be of the order of minutes, so schema resolution failures may cause a similar delay in the running of the XMLArtifactValidator listener.

Appendix B: Enabling a Listener

This example shows how a typical Listener is enabled using the LPC document. The example LPC process definition shown here configures an instance of the XMLArtifactValidator Listener class to be used for validating WSDL artifacts on web service assets. Only if the WSDL artifact on an asset is valid is it submitted for publishing. It assumes that there are no processes enabled or other customized listener behavior defined.

LPC Process Definition

```
<process-definition name="Asset Submission">
<custom-events>
        <custom-event>ASSET_VALIDATED</custom-event>
</custom-events>
    <enabled-processes></enabled-processes>
    <listeners>
        <listener name="AssetSubmission"
                        class="AssetSubmissionListener"></listener>
        <listener name="WSDLValidator" class="XMLArtifactValidator">
            <properties>
                <property name="target-artifact-category" value="wsdl" />
                <property name="result-artifact-category"
                                        value="validation-report" />
                <property name="locking-user" value="support" />
            </properties>
        </listener>
    </listeners>
    <filters>
        <filter name="WebServiceSubmissions">
            <event>ASSET_SUBMISSION_APPROVED</event>
            <classification-criteria-sets>
                <classification-criteria-set-name>Web service Assets</classification-criteria-set-
name>
            </classification-criteria-sets>
        </filter>
        <filter name="OtherSubmissions">
            <event>ASSET SUBMISSION APPROVED</event>
            <classification-criteria-sets complement="true">
                <classification-criteria-set-name>Web service Assets</classification-criteria-set-
name>
            </classification-criteria-sets>
        </filter>
    </filters>
    <actions>
        <action name="WebServiceValidation">
            <trigger-event>
                <event-filter>WebServiceSubmissions</event-filter>
            </trigger-event>
            <listener>WSDLValidator</listener>
            <result-event event="ASSET VALIDATED">
                <result-condition>0</result-condition>
            </result-event>
        </action>
        <action name="DefaultValidation">
            <trigger-event>
                <event-filter>OtherSubmissions</event-filter>
            </trigger-event>
            <result-event event="ASSET_VALIDATED"></result-event>
        </action>
        <action name="Submit Asset">
            <trigger-event>
                <event>ASSET_ VALIDATED</event>
            </trigger-event>
            <listener>AssetSubmission</listener>
        </action>
    </actions>
</process-definition>
```

Notes

Custom Event
This process flow declares a custom event called *ASSET_ VALIDATED.* The use of this event is discussed in the Action notes below.

Listener
An instance of the internal Listener class XMLArtifactValidator is defined and named "WSDLValidator". Two properties are provided to the Listener corresponding to the properties required for the XMLArtifactValidator class as specified in Appendix A. The first property indicates that artifacts of the category "wsdl" should be validated. The second property specifies the category that validation results may be added to[38].

Filter
A Filter named **"**WebServiceSubmissions" is defined to accept Events corresponding to the submission for publish of Assets that comply with the Classification Criteria Set "Web service Assets". This Classification Criteria Set must have been previously defined in the Library that this LPC document will be deployed to. The "Web service Assets" Classification Criteria Set will likely filter by asset-type, specifying types of "Web service" and "Web service Deployment".
A second filter named "OtherSubmissions" is defined to accept asset submission events for all other types of assets. This is done by specifying the "complement" attribute as true in the <classification-criteria-sets> element.

Action
The **"**WebServiceValidation" Action is defined to trigger the "WSDLValidator" Listener when events matching the "WebServiceSubmissions" Filter occur. This action will raise the custom event *ASSET_ VALIDATED* upon successful completion of the WSDL validation.
The "DefaultValidation" Action is used to map events matching the "OtherSubmissions" filter to the *ASSET_VALIDATED* event. This effectively causes all submitted Assets that are not web services to be considered validated.
Finally, the "Submit Asset" Action has been changed to trigger off to the *ASSET_VALIDATED* custom event[39].

Appendix C: A Simple Process Example

This appendix shows the enabling of a simple process for asset deletion that requires the approval of an asset owner. It assumes that there are no other processes enabled or customized listener behavior defined.

[38] It is assumed that both the "wsdl" category and the results category "wsdl-validation-results" have been defined in the Global Definition Template.
[39] Note that an alternative approach would have been to omit the "DefaultValidation" Action and trigger the "Asset Submit" action from either the *ASSET_VALIDATED* event *or* the "OtherSubmissions" Filter directly.

LPC Process Definition

```
<process-definition name="Asset Deletion">
    <custom-events></custom-events>
    <enabled-processes>
        <process>
            <name>ASSET_DELETION</name>
        </process>
    </enabled-processes>
    <listeners>
        <listener name="AssetDeletion" class="AssetDeletionListener"></listener>
        <listener name="DeletionAssetOwnerNotification"
            class="GenericRequestHandler">
            <properties>
                <property name="request-type" value="ASSET_DELETION" />
                <property name="request-state"
                                    value="Pending Asset Owner Approval" />
                <property name="recipient-role" value="Asset Owner" />
                <property name="recipient-message-id"
                    value="APPROVER_ACTION_REQUIRED" />
                <property name="lock-asset" value="true"></property>
                <property name="locking-user" value="support" />
            </properties>
        </listener>
        <listener name="DeletionRequestApproval"
            class="GenericRequestHandler">
            <properties>
                <property name="request-type" value="ASSET DELETION"></property>
                <property name="request-state" value="Approved" />
                <property name="terminate-request" value="true" />
                <property name="submitter-message-id"
                    value="SUBMITTER_REQUEST_APPROVED" />
                <property name="lock-asset" value="false" />
            </properties>
        </listener>
        <listener name="DeletionRequestRejection"
            class="GenericRequestHandler">
            <properties>
                <property name="request-type" value="ASSET_DELETION" />
                <property name="request-state" value="Rejected" />
                <property name="terminate-request" value="true" />
                <property name="submitter-message-id"
                    value="SUBMITTER_REQUEST_REJECTED" />
                <property name="lock-asset" value="false" />
            </properties>
        </listener>
    </listeners>
    <filters></filters>
    <actions>
        <action name="NotifyAssetOwnerForDeletion">
            <trigger-event>
                <event>ASSET_DELETION_REQUESTED</event>
            </trigger-event>
            <listener>DeletionAssetOwnerNotification</listener>
        </action>
        <action name="RejectDeletionRequest">
            <trigger-event>
                <event>ASSET_DELETION_Asset Owner_REJECTED</event>
            </trigger-event>
            <listener>DeletionRequestRejection</listener>
        </action>
        <action name="ApproveDeletionRequest">
            <trigger-event>
                <event>ASSET_DELETION_Asset Owner_APPROVED</event>
            </trigger-event>
            <listener>DeletionRequestApproval</listener>
            <result-event event="ASSET_DELETION_APPROVED" />
        </action>
        <action name="Delete Asset">
            <trigger-event>
```

```
            <event>ASSET_DELETION_APPROVED</event>
          </trigger-event>
          <listener>AssetDeletion</listener>
        </action>
    </actions>
</process-definition>
```

Notes

Process
First notice that the "ASSET_DELETION" process integration point has been activated:

```
<process>
   <name>ASSET_DELETION</name>
</process>
```

This is required in order to direct the Lifecycle Manager UI (in both web and IDE clients) to transfer the user to an Asset Request submission page upon requesting Asset deletion.

Listeners
Next notice that three Listener instances are defined using the GenericRequestHandler class in addition to the standard Listeners from the default LPC document. The first of these ("DeletionAssetOwnerNotification") is configured to be invoked when the requesting User submits the Asset Request for an Asset deletion. Its properties specify that the Asset Request should initially be placed in the state "Pending Asset Owner Approval" and that Asset Owners should be notified using the generic mail template. The *lock-asset* and *locking-user* properties specify that the Asset should be locked by the user "support" upon submission of the deletion request.

The second Listener ("DeletionRequestApproval") is configured to be run upon approval of the Asset Request by an Asset Owner. This instance of the GenericRequestHandler class is configured to move the Asset Request to the "Approved" state and terminate (inactivate) the Asset Request. The submitter of the request is notified of the approval using the appropriate generic mail template.

The third Listener ("DeletionRequestRejection") is configured to be run upon rejection of the Asset Request by an Asset Owner. This instance of the GenericRequestHandler class is configured to move the Asset Request to the "Rejected" state and terminate (inactivate) the Asset Request. The submitter of the request is notified of the rejection using the appropriate generic mail template.

Actions
The first Action defined ("NotifyAssetOwnerForDeletion") triggers the "DeletionAssetOwnerNotification" listener on the occurrence of the *ASSET_DELETION_REQUESTED* Event (the Event that is raised when the submitter submits the Asset for Asset deletion).

The next Action defined ("RejectDeletionRequest") is used to trigger the "DeletionRequestRejection" listener upon rejection of the Asset Request by an Asset Owner. Note that it is triggered by the Event *ASSET_DELETION_Asset Owner_REJECTED*, whose

name is formed dynamically from the Asset Request type: “ASSET_DELETION”, the Asset Owner’s role: “Asset Owner”, and the action: “REJECTED”.

The third Action (“ApproveDeletionRequest”) is very similar to previous Action and invokes the “DeletionRequestApproval” Listener upon approval of the Asset Request by an Asset Owner. Notice however, that this Action specifies that the Result Event *ASSET_DELETION_APPROVED* be raised upon completion of the Action. This event is is used to trigger the “DeleteAsset” Action.

Finally, the “Delete Asset” Action triggers the “AssetDeletion” listener from the *DELETION_SUBMISSION_APPROVED* Event raised by the previous Action. The “AssetDeletion” Listener performs the actual task of deleting the Asset.

Notice that the “Submit Asset” Action is left in its original form from the default LPC document since only the Asset Deletion process is being customized.

Appendix D: A Process Example Involving Parallel Approvals

This library configuration is somewhat more complex than the example in Appendix C. It customizes the Asset submission process to require first the approval of an Asset Owner and then approvals by a security architect and a database architect depending on the type of the Asset involved. If the approvals of both architects are required they may occur in any order. However, the submission will not be granted final approval until both architects have approved. This example demonstrates the use of custom Group Roles, Filters, and synchronized Actions. This example assumes the following roles are defined in the <group-roles> element:

```
<group-role>SecurityArchitect</group-role>
<group-role>DatabaseArchitect</group-role>.
```

LPC Process Definition

```
<process-definition name="Asset Submission">
    <custom-events></custom-events>
    <group-roles>
        <group-role>SecurityArchitect</group-role>
        <group-role>DatabaseArchitect</group-role>
    </group-roles>
    <enabled-processes>
        <process>
            <name>ASSET_SUBMISSION</name>
        </process>
    </enabled-processes>
    <listeners>
        <listener name="AssetOwnerNotification"
            class="GenericRequestHandler">
            <properties>
                <property name="request-type" value="ASSET_SUBMISSION" />
                <property name="request-state"
                    value="Pending Asset Owner Approval"/>
                <property name="recipient-role" value="Asset Owner" />
                <property name="recipient-message-id"
                    value="APPROVER_ACTION_REQUIRED" />
                <property name="lock-asset" value="true"></property>
                <property name="locking-user" value="support" />
            </properties>
        </listener>
        <listener name="SecurityArchitectNotification"
            class="GenericRequestHandler">
            <properties>
```

```
                <property name="request-type" value="ASSET_SUBMISSION" />
                <property name="request-state"
                    value="Pending Architect Approvals" />
                <property name="recipient-role"
                    value="SecurityArchitect" />
                <property name="recipient-message-id"
                    value="APPROVER_ACTION_REQUIRED"/>
            </properties>
        </listener>
        <listener name="DatabaseArchitectNotification"
            class="GenericRequestHandler">
            <properties>
                <property name="request-type" value="ASSET_SUBMISSION" />
                <property name="request-state"
                    value="Pending Architect Approvals" />
                <property name="recipient-role"
                    value="DatabaseArchitect" />
                <property name="recipient-message-id"
                    value="APPROVER_ACTION_REQUIRED" />
            </properties>
        </listener>
        <listener name="SubmissionApproval"
            class="GenericRequestHandler">
            <properties>
                <property name="request-type"
                    value="ASSET_SUBMISSION"/>
                <property name="request-state" value="Approved" />
                <property name="terminate-request" value="true" />
                <property name="submitter-message-id"
                    value="SUBMITTER_REQUEST_APPROVED" />
                <property name="lock-asset" value="false" />
            </properties>
        </listener>
        <listener name="SubmissionRejection"
            class="GenericRequestHandler">
            <properties>
                <property name="request-type" value="ASSET_SUBMISSION" />
                <property name="request-state" value="Rejected" />
                <property name="terminate-request" value="true" />
                <property name="submitter-message-id"
                    value="SUBMITTER_REQUEST_REJECTED" />
                <property name="lock-asset" value="false" />
            </properties>
        </listener>
        <listener name="AssetDeletion" class="AssetDeletionListener"></listener>
        <listener name="AssetSubmission"
            class="AssetSubmissionListener">
        </listener>
    </listeners>
    <filters>
        <filter name="SecurityArchitectApprovalRequired">
            <event>ASSET SUBMISSION Asset Owner_APPROVED</event>
            <classification-criteria-sets>
                <classification-criteria-set-name>SecurityApplicableAssets</classification-
criteria-set-name>
            </classification-criteria-sets>
        </filter>
        <filter name="SecurityArchitectApprovalNotRequired">
            <event>ASSET_SUBMISSION_Asset Owner_APPROVED</event>
            <classification-criteria-sets complement="true">
                <classification-criteria-set-name>SecurityApplicableAssets</classification-
criteria-set-name>
            </classification-criteria-sets>
        </filter>
        <filter name="DatabaseArchitectApprovalRequired">
            <event>ASSET_SUBMISSION_Asset Owner_APPROVED</event>
            <classification-criteria-sets>
                <classification-criteria-set-name>DatabaseApplicableAssets</classification-
criteria-set-name>
            </classification-criteria-sets>
        </filter>
```

```
        <filter name="DatabaseArchitectApprovalNotRequired">
            <event>ASSET SUBMISSION Asset Owner APPROVED</event>
            <classification-criteria-sets complement="true">
                <classification-criteria-set-name>DatabaseApplicableAssets</classification-
criteria-set-name>
            </classification-criteria-sets>
        </filter>
    </filters>
    <actions>
        <action name="NotifyAssetOwner">
            <trigger-event>
                <event>ASSET_SUBMISSION_REQUESTED</event>
            </trigger-event>
            <listener>AssetOwnerNotification</listener>
        </action>
        <action name="NotifySecurityArchitect">
            <trigger-event>
                <event-filter>SecurityArchitectApprovalRequired</event-filter>
            </trigger-event>
            <listener>SecurityArchitectNotification</listener>
        </action>
        <action name="NotifyDatabaseArchitect">
            <trigger-event>
                <event-filter>DatabaseArchitectApprovalRequired</event-filter>
            </trigger-event>
            <listener>DatabaseArchitectNotification</listener>
        </action>
        <action name="BypassSecurityArchitectApproval">
            <trigger-event>
                <event-filter>SecurityArchitectApprovalNotRequired</event-filter>
            </trigger-event>
            <result-event
                event="ASSET_SUBMISSION_SecurityArchitect_APPROVED" />
        </action>
        <action name="BypassDatabaseArchitectApproval">
            <trigger-event>
                <event-filter>DatabaseArchitectApprovalNotRequired</event-filter>
            </trigger-event>
            <result-event
                event="ASSET_SUBMISSION_DatabaseArchitect_APPROVED" />
        </action>
        <action name="ApproveSubmission" type="SYNCHRONIZED">
            <trigger-event>
                <event>ASSET_SUBMISSION_SecurityArchitect_APPROVED</event>
            </trigger-event>
            <trigger-event>
                <event>ASSET_SUBMISSION_DatabaseArchitect_APPROVED</event>
            </trigger-event>
            <listener>SubmissionApproval</listener>
            <result-event event="ASSET_SUBMISSION_APPROVED" />
        </action>
        <action name="RejectSubmission">
            <trigger-event>
                <event>ASSET_SUBMISSION_Asset Owner_REJECTED</event>
            </trigger-event>
            <trigger-event>
                <event>ASSET_SUBMISSION_SecurityArchitect_REJECTED</event>
            </trigger-event>
            <trigger-event>
                <event>ASSET_SUBMISSION_DatabaseArchitect_REJECTED</event>
            </trigger-event>
            <listener>SubmissionRejection</listener>
        </action>
        <action name="SubmitForPublish">
            <trigger-event>
                <event>ASSET_SUBMISSION_APPROVED</event>
            </trigger-event>
            <listener>AssetSubmission</listener>
        </action>
        <action name="Delete Asset">
            <trigger-event>
```

```
            <event>ASSET_DELETION_APPROVED</event>
          </trigger-event>
          <listener>AssetDeletion</listener>
        </action>
    </actions>
</process-definition>
```

Notes

The notes for this example will not discuss the basic topics already covered in Appendix B, but will instead focus on the new concepts introduced in this example.

Group Roles

Since Lifecycle Manager does not by default define Security Architect and Database Architect roles, these roles are defined in the "group-roles" element as "SecurityArchitect" and "DatabaseArchitect" respectively. These roles will be used extensively in the configuration of Listeners and will occur in the generated Asset Request approval Events.

Listeners

As in the previous example, a number of instances of the GenericRequestHandler Listener class are configured to correspond with the approvals by each role-player involved in the process. Another instance is defined to handle the rejection (by any role-player) of the Asset Request. Yet another instance is defined to handle the final approval of the Asset Request.

Filters

Since this process is dependent on the type of Asset involved, a number of Filters have been defined.

The first Filter ("SecurityArchitectApprovalRequired") is configured to listen for the Event indicating Asset Owner approval of a submission request for an Asset that complies with the Classification Criteria Set (CCS) "SecurityApplicableAssets"[40].

The next Filter ("SecurityArchitectApprovalNotRequired") is logically the complement of the first. It is triggered by Asset Owner approval of submission requests for Assets that do not comply to the "SecurityApplicableAssets" CCS. This is accomplished through the use of the "complement" attribute in the classification-criteria-set element:

```
<classification-criteria-set name="SecurityApplicableAssets"
                complement="true" />
```

The last two Filters are similar to the first two but pertain to the DatabaseArchitect role.

Actions

The "NotifySecurityArchitect" and "NotifyDatabaseArchitect" Actions are used to trigger listeners configured to notify users of the appropriate Group roles when an Asset Owner approves an asset submission request that passes the associated Filter. Note that if an asset submission was approve by an Asset Owner that met the criteria of both the

[40] This example assumes that a Classification Criteria Set has been defined in the target Library that specifies assets that will require security architect approval. This is likely implemented by defining a dedicated classifier in the global definition template for this purpose and defining a CCS that selects Assets that have a value of "Security" for this classifier.

"SecurityArchitectApprovalRequired" and "DatabaseArchitectApprovalRequired" Filters, both Listeners would be run, notifying both SecurityArchitects and DatabaseArchitects.

The Actions "BypassSecurityArchitectApproval" and "BypassDatabaseArchitectApproval" do not specify Listeners, only a result Event. Their purpose is to convert Events matching their associated Filter to the specified result Event. In the case of "BypassSecurityArchitectApproval", Events matching the Filter "SecurityArchitectApprovalNotRequired" are converted to the Event *Asset Submission_SecurityArchitect_APPROVED*. Coupled with the earlier defined Actions, this ultimately causes Asset Owner approval of the submission of an Asset *that does not require SecurityArchitect approval* to automatically trigger the SecurityArchitect approval Event, making it simpler to define the subsequent "ApproveSubmission" Action.

The "ApproveSubmission" Action is used to trigger the final GenericRequestHandler instance ("SubmissionApproval") that will finalize the Asset Request. It will also trigger the result Event *ASSET_SUBMISSION_APPROVED* which will then trigger the default "SubmitForPublish" Action that actually submits the Asset for publishing. What is unique about the "ApproveSubmission" Action is that it is marked as "SYNCHRONIZED", indicating that both trigger Events (*Asset Submission_SecurityArchitect_APPROVED* and *Asset Submission_DatabaseArchitect_APPROVED*) must occur before the associated Listener is invoked. This allows the process to wait for approvals by both architects in the cases where both approvals are required.

As with the Appendix C example, rejection by any role-player terminates the Asset Request. This is accomplished by the "RejectSubmission" Action.

Appendix E: External Events

It is possible for an Event to originate from an external source and trigger Actions defined in the LPC document in the same fashion as an internal Event. This appendix describes the use of external events.

Declaring an External Event

External Events are considered "custom" Events and must be declared as such in the LPC document:

```
<custom-events>
      <custom-event>MY_EXTERNAL_EVENT</custom-event>
</custom-events>
```

Once declared, the Event can be used in the standard way on Filter and Action declarations.

Creating an External Event

The structure of the Event element is defined in the *LibraryAPI* WSDL document and also reflected in the com.logiclibrary.external.notification.Event class in the Lifecycle Manager *common.jar*[41].

Required parameters provided on Event creation are:

[41] See Lifecycle Manager SOAP API documentation for additional details on using the LibraryAPI interface and Lifecycle Manager client jars.

- *libraryId*
  The identifier for the target Lifecycle Manager library (available from the Lifecycle Manager "Support Center" page).
- *eventType*
  The name of the event. This must match the event name declared in the *custom-events* element of the LPC document.
- *severity*
  Indicates the severity of the Event. Recommended value for external Events is "INFO_PRIMARY".
- *category*
  Indicates the grouping the Event belongs in. External events should use the "CUSTOM" category.
- *component*
  Lifecycle Manager component the event pertains to. External events should use the "LIBRARY" component.

Optional parameters for Event creation are:

- *assetId*
  Unique ID of an asset specified in the case where the Event corresponds to a specific asset.
- *orgGroupName*
  Name of a Lifecycle Manager organizational Group (or Project) specified in the case where an Event corresponds to a specific Group.
- *userId*
  UserId of a Lifecycle Manager User specified in the case where an Event corresponds to a specific User.
- Properties
  Properties in the form of name/value pairs can be added to the Event. These properties are available to Listeners triggered by the Event.

Signaling an External Event

The *notifyListeners()* method provided by Lifecycle Manager in the *LibraryAPI* SOAP interface is used to signal the external Event to the Lifecycle Manager library configuration engine. The external Event is passed as the sole parameter to this method. Note that while the *notifyListeners* method is synchronous, the firing of Actions as a result of the Event being signaled is performed asynchronously.

Sample Java Client Code

The following code example uses the Lifecycle Manager client Java JARs to create and signal an external Event:

```
public static void signalExternalEvent() throws Exception {

    //get the ILibraryAPI adapter
    ILibraryAPI libraryAPI = getLibraryAPI();

    //specify the libraryId of the target library
    String libraryId = "12:34";
```

```
        //note that this event must be declared as a custom event in the LPC document
        String eventType = "MY_EXTERNAL_EVENT";

        //it's recommended that externally created events use "INFORMATIONAL_PRIMARY" for severity
        String severity = EventDefinition.SEVERITY_INFORMATIONAL_PRIMARY;

        //external events should use "CUSTOM" as the event category
        String category = EventDefinition.CUSTOM;

        //it's recommended that externally created events use "LIBRARY" as component
        String component = EventDefinition.LIBRARY_COMPONENT;

        //create the Event
        Event externalEvent = new Event(libraryId, eventType, severity, category, component);

        //if this Event pertained to a particular asset, Group, or User we would set the appropriate
        //attributes on the event using the Event methods setAssetId, setOrgGroupName, and setUserId

        //add properties to the event.  Note that this property will be available to Listeners triggered
        //by the Event so it is a convenient method of propagating arbitrary contextual information from Event creator
        //to the triggered Listeners
        Properties eventProperties = new Properties();
        eventProperties.put("my_first_property","first_property_value");
        eventProperties.put("my_second_property","second_property_value");
        externalEvent.setProperties(eventProperties);

        //now signal the event with the Lifecycle Manager library configuration engine
        libraryAPI.notifyListeners(externalEvent);
    }

    protected static ILibraryAPI getLibraryAPI() throws Exception {
        //create a properties object containing necessary properties for the LibraryAPIClientAccess
        //factory method
        Properties properties = new Properties();
        //add the qualified class name of the adapter class that will be used
        properties.put(
            LibraryConstants.LIBRARYAPI_ADAPTER_CLASS_PROPERTY,
            "com.logiclibrary.library.client.LibraryAPISOAPAdapter");
        //add the qualified URL of the Lifecycle Manager server
        properties.put(
            AssetSourceConstants.URL_PROPERTY,"http://MyLifecycle ManagerServer/Lifecycle Manager/");
        //add the authenticator instance (created previously)
        properties.put(com.logiclibrary.assetsource.AssetSourceInternalConstants.AUTHENTICATOR,
                Authenticator.getAuthenticator(null));

        //invoke the factory method with the properties to obtain an initialized adapter instance
        return LibraryAPIClientAccess.getLibraryAPI(properties);
    }

    //this method initializes the global Authenticator instance used in getLibraryAPI()
        protected static void initializeAuthenticator() throws Exception {
        //create a properties object containing necessary properties for initializing an Authenticator instance
        Properties authprops = new Properties();
        //add the qualified class name of the Authenticator class that will be used
        authprops.put(Authenticator.AUTHENTICATOR, "com.logiclibrary.authentication.LDAPAuthenticator");
        //add the userId to be authenticated
        authprops.put(Authenticator.USERID, “support”);
        //add the password for authentication
        authprops.put(Authenticator.PASSWORD, “support_password”);
        //add the name of the library to authenticate to
        authprops.put(Authenticator.LIBRARYNAME,”targetLibraryName”);
        //add the URL of the Lifecycle Manager server
        authprops.put(Authenticator.AUTHURL, ,"http://MyLifecycle ManagerServer”);
        //create an instance of an authenticator
        Authenticator.initialize(authprops);
    }
```

Appendix F: A Sample Timer Application

This appendix describes how a Timer can be used in conjunction with a scripted listener to run an update script nightly in "batch" style.

Define a Custom Event

The first step is to define a custom event that can be raised by our Timer to trigger the RunCommand Listener. While the actual name of this event is arbitrary, this example will use "BATCH_UPDATE_TRIGGER". Define the custom Event as shown:

```
<custom-event>BATCH_UPDATE_TRIGGER</custom-event>
```

Define a Timer

Next, we'll configure a Timer to raise the "BATCH_UPDATE_TRIGGER" event nightly at 1:00AM:

```
<timer name="BatchUpdateTimer">
   <!-- Interval of one day -->
   <interval>1440</interval>
   <event> BATCH_UPDATE_TRIGGER </event>
   <start-time>2005-10-18T01:00:00</start-time>
   <firing-window>120</firing-window>
</timer>
```

We have used "BatchUpdateTimer" for the timer name, set the interval to 1440 minutes (24 hours) and set the initial start time to 1:00AM on October 18th, 2005. The timer is configured to raise the custom Event we defined in the previous step. This Timer definition also specifies a 120 minute (two hour) firing window to ensure that it will only fire within two hours of 1:00AM[42]

Define a Listener

An instance of ScriptListener is now defined in the <listeners> element that will run the custom update script:

```
<listener name="BatchUpdate" class="ScriptListener">
      <properties>
   <property name="script-id"
             value="BatchUpdateScript.bsh" />
</properties>
</listener>
```

The Listener is given the name "BatchUpdate" and "ScriptListener" as the Listener class. The document id of the batch script is provided in the property "script-id"[43].

[42] This avoids the possibility of a server outage pushing the firing time into busy daytime hours.

[43] Its assumed this script has been added to the Document Repository (see Appendix O and ScriptListener for additional information on script support)..

Define an Action

Finally, an Action is defined to trigger the "BatchUpdate" Listener from the Timer event "LDAP_RESYNC":

```
<action name="Batch Update Action">
   <trigger-event>
      <event>BATCH_UPDATE_TRIGGER</event>
   </trigger-event>
   <listener>BatchUpdate</listener>
</action>
```

Appendix G: Example Role/View Configuration for Restricted Access

This example shows how metadata-groups, metadata-views and Group roles may be used to restrict the visibility of a set of Asset elements to users of a particular Group role.

LPC Snippet

```
<metadata-groups default-ordering="ALPHA">
   <metadata-group name="Restricted Classifiers">
      <classifier name="business-value" />
      <classifier name="estimated-cost" />
   </metadata-group>
</metadata-groups>
<metadata-views>
   <metadata-view name="Default">
      <metadata-group name="Restricted Classifiers"
                                 complement="true" />
   </metadata-view>
   <metadata-view name="Restricted View">
      <metadata-group name="ALL_ELEMENTS" />
   </metadata-view>
</metadata-views>
<group-roles default-metadata-view="Default">
   <group-role name="Architect">
      <metadata-view name="Restricted View" />
   </group-role>
</group-roles>
```

Notes

In this example, the metadata-group "Restricted Classifiers" is defined to contain the classifiers that will be restricted from standard users. The "Default" metadata-view (the view that standard users will have) is set to the complement of the "Restricted Classifiers" metadata-group, meaning that the view contains all other possible Asset elements. The "Restricted View" metadata-view is then configured to have all possible Asset elements (including those specified in the "Restricted Classifiers" metadata-group). Finally, the "Architect Role" is defined and given access to the "Restricted View" metadata-view.

The result of this configuration is that only users having the "Architect" role on the active Project will be able to see the "business-value" and "estimated-cost" classifiers.

Appendix H: Artifact Sources

ClearCaseArtifactSource

- **Behavior:**
  The ClearCaseArtifactSource functions as a Rational ClearCase client retrieving ClearCase content. By using the ClearCaseArtifactSource, Lifecycle Manager browser and rich client interfaces are able to access files stored in ClearCase. The artifact reference takes the following format:

  soa://<cc>[/pvob:<*pvob*>][/baseline:<*baseline*>][/component:<*component*>]/fpath:<*fpath*>

  Where:
  - *pvob*
    The project repository managing the file for UCM configured FederatedSystems, otherwise the vob for Base ClearCase FederatedSystems.
  - *Baseline (UCM only)*
    The baseline identifying the proper version of the file.
  - *component (UCM only)*
    The component containing the file. If the component is a composite, the non-composite component should be used in this field. The composite component's baseline must be identified in the baseline field.
  - *fpath*
    The file path.

  If any of the optional values are missing from the URL, the artifact source will defer to the values defined in the classifiers specified by the artifact source properties. If the pvob is not found in the asset, the pvob defined on the ClearCaseSystem, identified by the federated-system-name property, will be used.
- **Usage Context:**
  Useful in retrieving artifacts from Rational ClearCase.
- **System Requirements:**
  The Lifecycle Manager application must have a supporting ClearCaseSystem configured. The ClearCaseSystem artifact-source-name property must match the name attribute specified in the artifact-source definition element of the LPC.
- **Properties:**
  - *federated-system-name*
    (Required) The name of the federated system that the artifact source represents
  - *pvob-default*
    The classifier which will specify the default pvob (UCM) or vob (Base ClearCase)
  - *baseline-default*
    (UCM only) The classifier which will specify the default baseline
  - *component-default*

(UCM only) The classifier which will specify the default component

- **Example LPC Entry:**

```
<artifact-source name="cc"
class="com.logiclibrary.artifact.sources.ClearCaseArtifactSource">
 <properties>
   <property name="federated-system-name" value="ClearCaseSystem"/>
   <property name="pvob-default" value="cc-pvob"/>
   <property name="baseline-default" value="cc-baseline"/>
   <property name="component-default" value="cc-component"/>
 </properties>
</artifact-source>
```

- **Suggested Classifier and Artifact Definitions/Usage:**

```
<define-classifier name="cc-pvob" display-name="ClearCase PVOB"
   type="string" max-occurs="1"
   help-text="ClearCase PVOB that applies to the asset (ex: /my_pvob)"/>
<define-classifier name="cc-baseline" display-name="ClearCase Baseline"
   type="string" max-occurs="1"
   help-text="ClearCase baseline that applies to the asset"/>
<define-classifier name="cc-component" display-name="ClearCase Component"
   type="string" max-occurs="1"
   help-text="ClearCase component that applies to the asset"/>
<define-artifact category="cc-reference"
   display-name="ClearCase Reference"
   help-text="A Lifecycle Manager reference to a ClearCase controlled
artifact."/>
<use-artifact category="cc-reference" default-type="by-reference"
   default-reference="soa://cc[[[/pvob:&lt;pvob&gt;]/baseline:&lt;
   baseline&gt;]/component:&lt;component&gt;]/fpath:&lt;fpath&gt;"/>
```

- **Example Artifact reference:**
  soa://cc/pvob:dfsproddev_pvob/baseline:prod_project_01_24_2008/component:helloworld/fpath:helloworld.jar

HTTPSource

- **Behavior:**
  Retrieves artifact contents from a web server requiring basic HTTP authentication.
- **Usage Context:**
  Useful in retrieving artifacts from an SCM system or other authenticated system of record that provides a web interface.
- **Properties:**
  - *url*
    The root URL to use in accessing artifact files. This property may specify a varying amount of path information depending on the focus of a particular HTTPSource instance.
  - *user*
    User name to use in HTTP authentication.
  - *Password*
    Password to use in HTTP authentication.
- **Example Artifact reference:**
  soa://p4/common/main/services/CurrencyExchange.wsdl
  Where /common/main/services/CurrencyExchange.wsdl is concatenated to the specified url property.

ExternalSOAPArtifactSource

- **Behavior:**
  Similar in concept to an External Listener, this ArtifactSource class retrieves artifact contents from an externally provided Web service.
- **Usage Context:**
  Useful in retrieving artifact content requiring custom logic to retrieve or assemble.
- **Properties:**
  - *accesspoint_url*
    The URL to of the external ArtifactSource service.
  - *auth_userid*
    User id to use in authenticating with the external ArtifactSource service.
  - *auth_password*
    Password to use in authenticating with the external ArtifactSource service.
  - *Additional Properties*
    Additional properties provided in the ArtifactSource definition in the LPC will be passed through to the external web service on the getArtifact and getArtifactInfo calls.

Notes
The target Web service must implement the ExternalArtifactSource wsdl found in the soapgen Lifecycle Manager client jar.

Please contact SOA Professional Services for guidance in providing an external ArtifactSource.

RepositoryManagerArtifactSource

- **Behavior:**
  Retrieves an artifact from the current asset matching a name specified either by reference or by the artifact-name property. This Artifact Source serves as a target for Artifact Transforms that run on existing artifacts within the asset.
- **Usage Context:**
  Used as a target for Artifact Transforms. Commonly used as a source for the XSLTransform Artifact Transform.
- **Properties:**
  - *artifact-name*
    Default artifact name of the artifact to retrieve. If not specified, the <key> portion of the custom artifact reference will be used as the artifact name.
- **Example Artifact reference:**
  soa://wsdlviewer/Service Interface and Implementation Definition
  Where wsdlviewer is the Artifact Source name and "Service Interface and Implementation Definition" will be used by the Lifecycle ManagerArtifactSource as the name of the target artifact to retrieve.

RTCArtifactSource

- **Behavior:**
  Retrieves an artifact from a Rational Team Concert server.
- **Usage Context:**
  Used to retrieve a RTC artifact without a user needing to authenticate or have access to a RTC server.
- **Properties:**
  - *server*
    This property should be set to the location of the RTC server Configuration and Change Management root (/ccm). The URL should resemble https://servername:9443/ccm.
  - *username*
    The user that the artifact source uses to connect to and retrieve the file contents
  - *password*
    The corresponding password for the user above.
- The URL used instructs the Artifact Source which version to retrieve from the server. There are certain parameters that can be added to the URL to specify a different version of the file, while some parameters are required. The general format for the URL is "soa://rtc/path/to/file?component=ComponentName&*versiondetails*". See the query parameters listed below for specific formatting information. If more than one version specification is used, the version used is undefined.
- **Query parameters**
  - *component*
    This is the component in the repository that the file resides in. The file reference is not valid without a component.
  - *snapshot*
    If a snapshot parameter is specified, and exists for the component, the versioned file in the snapshot will be returned.
  - *baseline*
    If a baseline is specified and corresponds to the component, the versioned file in the baseline will be returned.
  - *workspace*
    The file will be returned that exists in the workspace for the specified component

- **Example Artifact references:**
  soa://rtc/ProjectName/file.txt?component=ComponentName&baseline=compRelease1
  soa://tfs/ProjectName/file.txt?component=ComponentName&workspace=Example%20Stream

  The examples above retrieve different versions of ProjectName/file.txt in the component ComponentName. In the first instance, it uses a baseline named "compRelease1". In the second instance it uses workspace "Example Stream".

TFSArtifactSource

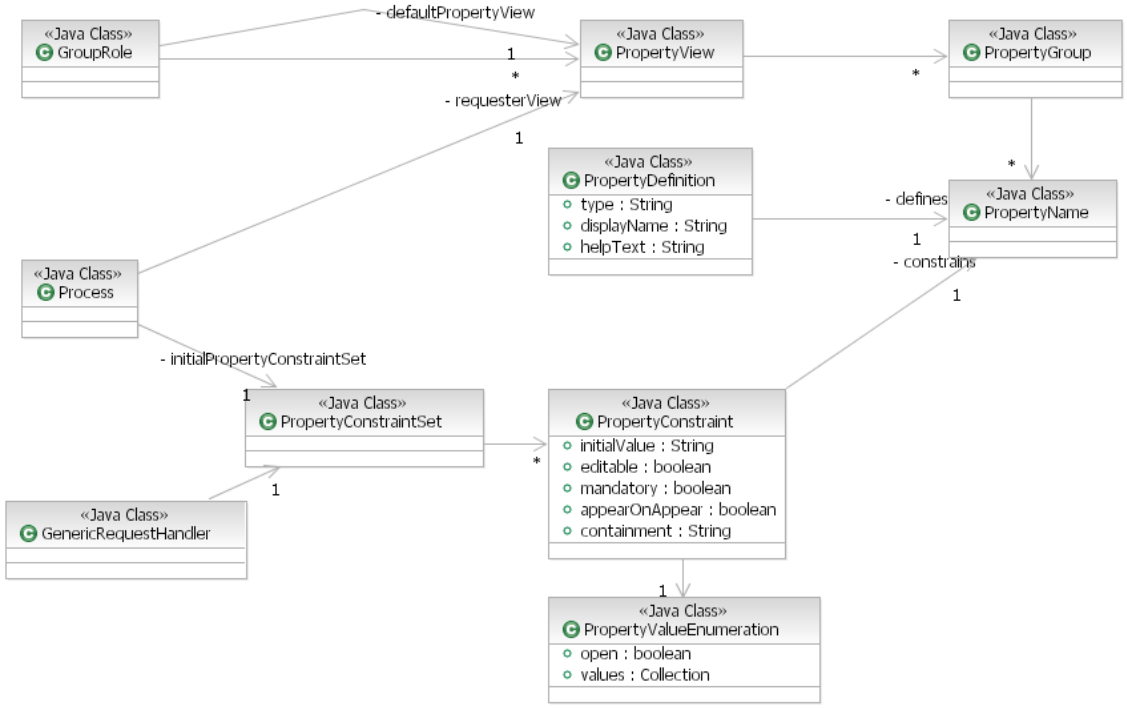
- **Behavior:**
  Retrieves an artifact from a Microsoft Team Foundation Server.
- **Usage Context:**
  Used to retrieve a TFS artifact without a user needing to authenticate or have access to a TFS server.
- **Properties:**
  - *server*
    This property should be set to the location of the TFS server root. For TFS 2005 and 2008, this should resemble: http://servername:8080/tfs. For TFS 2010, the URL should include the collection: http://servername:8080/tfs/FirstCollection.
  - *username*
    The user that the artifact source uses to connect to and retrieve the file contents
  - *password*
    The corresponding password for the user above.
  - *version*
    This property should be set to "2010" if the server is TFS 2010. If it is not set it will default to 2005 and 2008 support
- The URL used instructs the Artifact Source which version to retrieve the from the server. There are certain parameters that can be added to the URL to specify a different version of the file. The general format for the URL is "soa://tfs/path/to/file[;versionspec]?queryparameters". See the versionspec query parameter listed below for specific formatting information. If more than one version specification is used, the version used is undefined.
- **Query parameters**
  - *changeset*
    A integer representing the changeset containing the version of the file to return
  - *date*
    The version used is the most recent checked in version prior to the date. The format for the date should be "yyyy-mm-dd" or "yyyy-mm-dd hh:mm:ss" or "yyyy-mm-dd hh:mm:ss±tt:tt", where "±tt:tt" is the timezone offset. If a time isn't specified it defaults to 00:00 GMT. If a timezone isn't specified, it defaults to GMT.
  - *label*
    A value in the format "label@scope" representing the label containing the version of the file to return. The scope is usually not specified as it is inferred from the file path.
  - *versionspec*
    Microsoft's specific short naming convention for referring to a variety of version specifications. It is of the format: "C*changesetnumber*", "L*labelname*", "D*date*", "W*workspace;owner*", or "T' for the latest version. See the corresponding query parameters for more information.
  - *workspace*
    A value in "*workspace*;*owner*" that corresponds to the workspace version the user currently has. Caution: using the workspace parameter can lead to odd behavior unless workspace versions are controlled.

- **Example Artifact references:**
  soa://tfs/ProjectName/file.txt?label=1234
  soa://tfs/ProjectName/file.txt;C34
  soa://tfs/ProjectName/file.txt?date=2010-12-20
- soa://tfs/ProjectName/file.txt?versionspec=Lqa

  The examples above retrieve different versions of file.txt. In the first instance, it uses a label named "1234". In the second instance it uses changeset 34. In the third example, it uses the most recent version prior to the December 20, 2010. In the fourth example, it retrieves the version on the "qa" label.

Appendix I: Asset Request Property Configuration

The following diagram provides a visual representation of the overall property configuration structure in the LPC document (note that the relationships to Process and GenericRequestHandler apply only to Asset Request properties):



Property Definition

Static characteristics of Asset Request properties are defined within the <property-definitions> section of the LPC document using <property-definition> elements. A <property-definition> element allows the following to be specified:

- *name*
  Name (key) of the property being defined

- *target*
  This indicates the type of business object the Property Definition is applicable for. Choices for this field are:
  - *ASSET_REQUEST* – An Asset Request (this is the default target)
  - *CATALOG_ASSET* – The catalog version of an asset
  - *LIBRARY_ASSET* – The published version of an asset
  - *ASSET* – Applicable to both catalog and library versions of an asset.
  - *GROUP – An organizational group.*
  - *USER – A user object.*
- *type*
  *D*ata type of the property value. Choices for this field are:
  - *STRING*
  - *BOOLEAN*
  - *DECIMAL*
  - *DATE*
  - *FILE*
- *help-text*
  Used for hover text in the user interface
- *display-name*
  Actual Property name displayed in the user interface
- *value-source*
  The name of a Value Source instance to retrieve valid values from. See Value Sources.
- *copy-prohibited*
  In the case of Asset Request properties, it is possible to "prime" a new request from an existing one by copying the source request's properties to the new request. This property allows the LPC author to prohibit copying of certain properties (for example a "design review completed" property). Valid values for this property are "true" and "false". The attribute is optional and defaults to "false" (indicating that the property will be copied to a new request).

Here is a sample <property-definitions> section:

```
<property-definitions>
   <property-definition name="intended-usage"
      type="STRING"
      display-name="Intended Usage"
      help-text="Domain the asset will be used in."/>
   <property-definition name="support-committed"
      type="BOOLEAN"
      display-name="Support Committed"
      help-text="Has owner committed to support usage"
               copy-prohibited="true"/>
   <property-definition name="support-priority"
      type="STRING"
      display-name="Support Priority"/>
   <property-definition name="support-start-date"
      type="DATE"
      display-name="Support Start Date"/>
   <property-definition name="usage-contract"
      type="FILE"
```

```
            display-name="Usage Contract"/>
        <property-definition name="reuse-guidelines"
            type="FILE"
            display-name="Reuse Guidelines"/>
        <property-definition name="acceptance-testing-complete"
            type="BOOLEAN"
            display-name="Acceptance Testing Completed"
                    copy-prohibited="true"/>
        <property-definition name="acceptance-checklist"
            type="FILE"
            display-name="Acceptance Checklist"
                    copy-prohibited="true"/>
        <property-definition name="external-ref"
            type="FILE"
            display-name="External SOR Reference"
                    target="ASSET"/>
</property-definitions>
```

File Properties
As noted above, a property may have a type of "file" meaning that it is referring to a document. Such properties may contain by-value or by-reference documents.

Property Constraints

A <property-constraint> element is used to constrain the semantics of a defined property for particular request states. Property constraint elements are grouped into a uniquely named <property-constraint-set> which resides in the <property-constraint-sets> section of the LPC document. Process elements may optionally specify an initial <property-constraint-set> for the initial state of the request. Additionally the GenericRequestHandler Listener will allow a <property-constraint-set> to be specified for the state transition that the listener instance represents.

Property Constraint elements contain the following fields:

- *name*
  Name (key) of the property being constrained (mandatory).
- *Target*
  Target of the property being constrained. See Property Definition target for choices.
- *default-value*
  An initial value to be assigned to the property when the <property-constraint-set> is applied. This element may be further qualified with the attribute *override* which determines whether an existing property value will be overridden.
- *editable*
  Determines whether the value in the specified property is editable. The default setting is "true".
- *mandatory*
  Determines whether the user must set a value for the specified property. The default setting is "false".
- *enumeration*
  A sub-element used to specify a collection of possible values for an enumerated (string) property.

- *open*
  This is an attribute of the *enumeration* element that determines whether the user may specify a value other than the specified value set. Default value is true.
- *containment*
  In the case of a file property this attribute may be used to specify whether containment of the document is by-reference,by-value or both. Default value is both.

Note that only those defined properties that appear within the constraint-set currently associated with an Asset Request will be visible (regardless of role-based visibility). For this reason, a <property-constraint> element need only contain a name attribute in the case where a property is simply being made visible with no constraints.

If no constraint-set is associated with the Asset Request, all defined properties will be visible subject to role-based visibility constraints.

Here is a sample <property-constraint-set> element:

```
<property-constraint-set name="Pending AssetOwner Approval">
   <property-constraint name="intended-usage" editable="false">
   </property-constraint>
   <property-constraint name="support-committed">
      <default-value value="false" override="false" />
   </property-constraint>
   <property-constraint name="support-priority" mandatory="true">
      <enumeration open="false">
         <value>Low</value>
         <value>Medium</value>
         <value>High</value>
      </enumeration>
   </property-constraint>
   <property-constraint name="usage-contract"
      containment="BY_VALUE">
      <default-value
         value="http://abc.com/ReusecontractSkeleton.doc" />
   </property-constraint>
   <property-constraint name="external-ref"
      target="ASSET"
      containment="BY_REFERENCE">
   </property-constraint>
 </property-constraint-set>
```

Role-based Visibility

Request and Asset Property visibility is managed following the view/group pattern used for Asset metadata visibility. LPC sections <property-groups> and <property-views> contain <property-group> and <property-view> elements respectively. A <group-role> element may contain one or more <property-view> elements. The <group-roles> element may contain a "default-property-view" attribute that specifies the property-view used if a role has no property-views assigned. As with metadata-groups, an "ALL_PROPERTIES" virtual group is supported that makes all properties visible[44].

[44] Visibility is also subject to the current property-constraint-set as discussed earlier.

Here is a sample configuration:

```
<property-groups>
   <property-group name="Common Properties">
      <property name="reuse-guidelines" />
   </property-group>
   <property-group name="Producer Properties">
      <property name="support-committed" />
      <property name="support-priority" />
      <property name="support-start-date" />
      <property name="usage-contract" />
   </property-group>
   <property-group name="Consumer Properties">
      <property name="acceptance-testing-completed" />
      <property name="acceptance-checklist" />
   </property-group>
</property-groups>
<property-views>
   <property-view name="Asset Owner Request View">
      <property-group name="Common Properties" />
      <property-group   name="Producer Properties" />
   </property-view>
   <property-view name="Project Manager Request View">
      <property-group name="Common Properties" />
      <property-group   name="Consumer Properties" />
   </property-view>
   <property-view name="Common Request View">
      <property-group name="Common Properties" />
   </property-view>
</property-views>
```

And the group-roles section:

```
<group-roles
   default-metadata-view="Default Asset View"
   default-property-view="Common Request View">
   <group-role name="Asset Owner">
      <metadata-view name="Asset Owner Asset View" />
      <property-view name="Asset Owner Request View" />
   </group-role>
   <group-role name="Project Manager">
      <metadata-view name="Project Manager Asset View" />
      <property-view name="Project Manager Request View" />
   </group-role>
</group-roles>
```

Requester Views (Asset Requests)

Since there are no explicit roles assigned to the requester of an approval process, the <process> element may specify the property view to be used for the requester of that process. If this element is not specified, the requester will use the default Property View assigned in the <group-roles> element. Here is an example process definition specifying the initial Property Constraint set and requester view:

```
<process>
   <name>ASSET_ACQUISITION</name>
   <initial-property-constraint-set>Acquisition Requester
      </initial-property-constraint-set>
```

```
   <requester-property-view>Acquisition Requester
      </requester-property-view>
</process>
```

Role Selection (Library Assets)

When viewing a published asset, the properties displayed will be subjected to the group-role selection (similar to asset metadata elements). This means that if no group-role is selected, all properties accessible from the user's group-roles will be displayed. If a group-role is selected, only those properties in the property-views for that group-role will be displayed.

Filtered Process Selection

It may be desirable to select different initial-property-constraint-set and requester-property-views for different types of Assets or other context criteria. To facilitate this, the LPC document allows multiple <process> definitions for the same process (e.g. ASSET_ACQUISITION) to be defined. Care should be taken to specify mutually exclusive <filter> elements for each such process definition. In the case where a request action matches multiple Processes, the first process definition with a matching filter will be selected. Here is an example of filter based process definitions that vary the initial-property-constraint-set and requester-property-view by Asset type:

```
<process>
      <name>ASSET_ACQUISITION</name>
   <filter>
      <asset-filters>
         <asset-filter-name>Service Assets</asset-filter-name>
      </asset-filters>
   </filter>
      <initial-property-constraint-set>Acquisition Requester
Service</initial-property-constraint-set>
   <requester-property-view>Acquisition Requester Service</requester-
property-view>
</process>
<process>
   <name>ASSET_ACQUISITION</name>
   <filter>
      <asset-filters>
         <asset-filter-name>Non-service Assets</asset-filter-name>
      </asset-filters>
   </filter>
   <initial-property-constraint-set>Acquisition Requester Standard</initial-
property-constraint-set>
   <requester-property-view>Acquisition Requester Standard</requester-
property-view>
</process>
```

Ordering

Ordering of properties for display is similar to what is described for Asset metadata elements. Property ordering is determined by the ordering of properties within group roles, then Property Views, and finally within Property Groups.

Default Ordering

The <property-groups> element contains the optional attribute “default-ordering”. This attribute determines the ordering of Properties in the following cases:

- Ordering of Properties within the virtual “ALL_PROPERTIES” property-group
- Ordering of elements in the complement of a property-group

There are two choices for the value of the “default-ordering” attribute:

- “ALPHA” – Indicates that elements should be ordered alphabetically
- “DEFINITION” - Indicates that Properties should be ordered according to their definition in the property-definition section of the LPC.

Appendix J: Importers

The following Importer classes are currently available:

SchemaImporter

- **Behavior:**
  Creates Schema assets from an XML schema document. The SchemaImporter can also be configured to show relationships to dependent schemas (see the *create-dependent-relationships* property below) and also create any resources that do not exist as assets within Lifecycle Manager (see the *show-dependent-resources* property below).
- **Properties*:***
  - The Schema resolution properties are supported for this importer.
  - *description*
    The description of the importer.
    Defaults to “Import schema documents”. (optional)
  - *schema-artifact-containment*
    Determines the containment of the schema document artifact. Valid values are “by-value” and “by-reference”. This setting is only meaningful in the case where the artifact is accessed by URL; loading a schema document from a file will force artifact containment to be “by-value”.
    Defaults to “by-reference”. (optional)
  - *schema-asset-version*
    The asset version that will be used for assets created by this importer. Defaults to “1.0” (optional).
  - *schema-asset-type*
    The asset-type used for assets created by this importer.
    Defaults to “XML Schema” (optional).
  - *schema–asset-template*
    The capture template to use for assets created by this importer.
    Defaults to “XML Schema” (optional).
  - *schema-asset-description*
    The description used for assets created by this importer.
    Defaults to "Auto-generated XML Schema Asset" (optional).
  - *schema-asset-overview*
    The overview used for assets created by this importer.
    Defaults to "Auto-generated XML Schema Asset" (optional).

- *create-note*
  Note used for creation of schema Assets.
  Defaults to "Asset created by SchemaImporter" (optional).
- *submit-note*
  Note used for submission of schema Assets.
  Defaults to "Asset submitted by SchemaImporter" (optional).
- *adjust-schema-asset-name*
  If false, generated schema asset names will be the schema target namespace URI. If true and the target namespace is a valid URL, the path part of the URL will be used as the asset name with the "/" characters replaced with spaces. For example, if the target namespace URI is "http://schemas.xmlsoap.org/wsdl/soap/" and *adjust-schema-asset-name* is "true", the schema asset's name will be "wsdl soap". The default value for this property is "false". (optional)
- *show-dependent-resources*
  If this property is set to true, then dependent schemas will be displayed as potential assets for import. If assets with the same target namespace (as defined by the schema-namespace-classifier property) or the same location (as defined by the schema-location-classifier) already exist in the library, they will not be displayed.
- *create-dependent-relationships*
  If this property is set to true, then any referenced schemas will be shown in the asset's relationships. Related assets are determined by searching by either import/included namespace or location attributes (see schema-namespace-classifier or schema-location-classifier).
- *included-schema-relationship* – see *imported-schema-relationship*
- *imported-schema-relationship*
  If create-dependent-relationships classifier is set to true, the *included-schema-relationship* and *imported-schema-relationship* properties determine the names of the relationships that will be created in the imported asset. The *included-schema-relationship* and *imported-schema-relationship* will be created on any schema assets that reference other schemas.

- **Resulting Assets**
  A single asset of type specified by the importer properties will be created with the source schema document as an artifact. The schema's namespace and optionally location will be stored as classifiers on the asset.

WSDLImporter

- **Behavior:**
  Creates Web service assets from a WSDL document or a ZIP file containing WSDLs and their supporting schemas. If a ZIP file is used during import, it will be attached to every asset created from the ZIP. The WSDLImporter can also be configured to show relationships to dependent WSDLs and schemas(see the *create-dependent-relationships* property) and also create any resources that do not exist as assets within Lifecycle Manager (see the *show-dependent-resources* property). There are 4 possible asset types that can be created by the importer: Service Interface, Service Implementation, Service

Interface and Implementation, and Schemas. These asset types are based on the content of the files being processed (see the **-asset-type* and **-asset-template* properties).

- **Properties:**
  - The WSDL / schema resolution properties are supported for this importer.
  - *description*
    The description of the importer.
    Defaults to "Import WSDL documents". (optional)
  - *wsdl-artifact-containment*
    Determines the containment of the wsdl artifact. Valid values are "by-value" and "by-reference". This setting is only meaningful in the case where the artifact is uploaded from a URL; loading a WSDL document from a file will force artifact containment to be "by-value".
    Defaults to "by-reference". (optional)
  - *service-asset-version*
    The asset version that will be used for assets created by this importer. Defaults to "1.0" (optional).
  - *service-interface-asset-type*
    The asset-type used for service interface assets created by this importer. A service interface contains a WSDL that has no defined services.
    Defaults to "Service" (optional).
  - *service-implementation-asset-type*
    The asset-type used for service implementation assets created by this importer. A service implementation contains a WSDL that has defined services that imports another WSDL.
    Defaults to "Service Implementation" (optional).
  - *service-interface-and-implementation-asset-type*
    The asset-type used for implementation and interface assets created by this importer. A service interface contains a WSDL that has both defined services and does not import an interface WSDL.
    Defaults to "Service" (optional).
  - *service-interface-asset-template*
    The capture template to use for service interface assets created by this importer.
    Defaults to "Service Interface - Initial" (optional).
  - *service-implementation-asset-template*
    The capture template to use for service implementation assets created by this importer.
    Defaults to "Service Implementation - Initial" (optional).
  - *service-interface-and-implementation-asset-template*
    The capture template to use for service interface and implementation assets created by this importer.
    Defaults to "Service Interface and Implementation - Initial" (optional).
  - *service-asset-description*
    The description used for assets created by this importer.
    Defaults to the first service description in the WSDL if present, otherwise defaults to "Auto-generated Web Service Asset" (optional).

- *service-asset-overview*
  The overview used for assets created by this importer.
  Defaults to the first service description in the WSDL if present, otherwise defaults to "Auto-generated Web Service Asset" (optional).
- *create-note*
  Note used for creation of schema Assets.
  Defaults to "Asset created by WSDLImporter" (optional).
- *submit-note*
  Note used for submission of schema Assets.
  Defaults to "Asset submitted by WSDLImporter" (optional).
- *adjust-service-asset-name*
  If false and no service name attribute is found in the WSDL[45], generated schema asset names will be the WSDL target namespace URI. If true, no service name attribute is found in the WSDL, and the target namespace is a valid URL, the path part of the URL will be used as the asset name with the "/" characters replaced with spaces. For example, if the target namespace URI is "http://www.SOA.com/businesslogic/LibraryAPI/" and *adjust-schema-asset-name* is "true", the schema asset's name will be "businesslogic LibraryAPI". The default value for this property is "false". (optional)
- *show-dependent-resources*
  If this property is set to true, then dependent services and schemas will be displayed as potential assets for import. If assets with the same target namespace (as defined by the service-namespace-classifier property) or the same location (as defined by the service-location-classifier) already exist in the library, they will not be displayed.
- *create-dependent-relationships*
  If this property is set to true, then any referenced services and schemas will be shown in the asset's relationships. Related assets are determined by searching by either import/included namespace or location attributes (see service-namespace-classifier, schema-namespace-classifier, service-location-classifier, or schema-location-classifier).
- *referenced-service-relationship*
- *referenced-schema-relationship*
- *included-schema-relationship*
- *imported-schema-relationship*
  If create-dependent-relationships classifier is set to true, these properties determine the names of the relationships that will be created in the imported asset. The *referenced-service-relationship* and *referenced-schema-relationship* will be added to any service assets that reference other WSDLs or XSDs. The *included-schema-relationship* and *imported-schema-relationship* will be created on any schema assets that reference other schemas (if *show-dependent-resources* is set to true).
- *schema-asset-type*
- *schema-asset-template*

[45] Note that the "name" attribute value from a WSDL <service> element (if found) takes priority over the target namespace element.

- *schema-artifact-containment*
- These properties correspond to the *service-** properties and apply to and schema assets that are created as a result of the import. This is only applicable when the *show-dependent-resources* property is set to true.

- **Resulting Assets**
  A single asset of type specified by the importer properties will be created with the source WSDL document as an artifact. The schema's namespace will be stored as a classifier on the asset.

DelimitedFileAssetImporter

- **Behavior:**
  Imports assets specified in a delimited text file or a *.zip file. The zip file must contain an "assets.txt" delimited text file and any referenced by-value artifacts. The zip file can be manually created or created through the DelimitedFileAssetExporter. The importer format provides a means for a user to enter asset information quickly into a spreadsheet, save it as a delimited file, and load it into the library. The first row of the file must be asset attributes names. Subsequent lines contain the values of the attribute for an asset. The header may have repeated asset attributes to allow multiple values to be set on an attribute of an asset. Below is a list of the attribute names and their expected values.
  - *asset.id=<id of the asset> (Required only if updating pre-existing assets)*
  - *asset.name=<name of the asset>*
  - *asset.version=<version of the asset>*
  - *asset.template=<template>*
  - *asset.description=<description>*
  - *overview.text=<The overview text>*
  - *owning.group=<The name of the asset's owning group>*
  - *project.context=<The name of the asset's project context>*
  - *classifier.<name>=<value> (where <name> is the classifier name as specified in the template*
  - *artifact.<category>=<reference>^<by-reference|by-description>,<version>[^<name>]*
  - *relatedasset.<relation name>=<asset name>^<asset version>[^<property-name=property-value>[…]] (where <relation name> = predecessor | prerequisite | <a predefined relation name>*

  The following should be noted:
  - If asset.id is empty or not provided, the importer will inspect the library for the first asset with a matching name, version, and classifier.asset-type. If found, that asset will be replaced or updated (dependent on the 'replace' property), otherwise a new asset will be created in the library.
  - The artifact can be either by-reference or by-description. The <reference> value is one of the following:
    - by-value: The path to the artifact relative to the delimited file. Note that the file may be specified as `<filename>.<extension>`, `*.<extension>`, and `*.*`. The [^<name>] on the artifacts, if omitted, defaults to the category or, in the case of wildcarding, defaults to the file name (extension

included). The by-value artifacts and the delimited asset file (`assets.txt`) must be provided to the importer in the form of a zip file.
    - by-reference: The reference (typically a URL). [^<name>] on the artifacts, if omitted, defaults to the category.
    - by-description: The description. [^<name>] on the artifacts, if omitted, defaults to the category.
- If there are multiple artifacts on a row which share the same category, name, and reference (by-reference, by-description, or by-value), the first occurrence will update the matching artifact if found in the asset and the subsequent artifact occurrences will be added as new artifacts.
- The asset.publish file is only necessary if you are submitting the imported assets for publish and want to use non-default publish information.
- Depending on the volume of assets you are loading, you may want to consider:
    - Reducing the logging and/or altering the number of log files.
    - Temporarily remove contacts from the publish template.
    - Coordinate with your application administrators to disable email from the library.

- **Properties:**
    - *concurrentThread*
      The number of assets imports that may be simultaneously executed. (optional, default=1)
    - *loadFactor*
      A multiplier of concurrentThreads that controls the size of the importer queue. For example, setting concurrentThreads=3 and loadFactor=10.1 will allow the import queue to be loaded with 30.3 Asset import jobs. A higher loadFactor will allow the import UI to return quicker at the sacrifice of a larger import queue. (optional, default=10.0)
    - *attributeDelimiter*
      The delimiter character (default is tab). For example, the delimiter between classifiers, artifacts, etc
    - *attributeValueDelimiter*
      The attribute delimiter separating the values of the attributes (default '^'). Using an artifact as an example, this delimiter would separate the reference, by-reference, version, and optional name.
    - *thread-sleep*
      The milliseconds that an asset load task will wait before checking for an available thread. (optional, default=250)
    - *description*
      The importer description. (optional)
    - *replace*
      If true, each line in the delimited file will replace the asset found in the library. Setting this property to false will update assets found in the library with the contents of the delimited file. The update behavior will add[46] elements (ex: classifier, artifact, and relationship) which are not found in the asset by name and

[46] By-value artifacts are an exception to this rule. They will always replace an artifact matching by artifact name and category.

value[47]. If adding the element will exceed the number of allowable instances, the value from the delimited file will replace an existing element. NOTE: If *replace* is set to false and an asset is specified in multiple rows of the delimited file, *concurrentThread* should be set to 1. (optional, default=true)
    - *email-results*
      If true, the results of the import will be sent to the user. If false, the results will be placed in an "import_*.zip file found in the logging directory of the application and should be removed when appropriate. (optional, default=true)
    - *ignore-asset.id*
      If true, the importer will execute as if the asset.id is not specified in the data. See the note found in the Behavior section above. (optional, default=false)
- **Parameters:**
    - *Import File*
      The file to be imported. The file can take one of two forms: 1) An assets.txt delimited file containing all the information to create the assets but does not reference any by-value artifacts; 2) A *.zip file containing the delimited assets.txt file and by-value artifacts. (required)
    - *Require Validation*
      If true (default), requires that the asset passes validation prior to storing the asset in the catalog.
    - *Replace*
      Overrides the replace *property* configured in the lpc file.
    - *Submission Note*
      Submission request note
- **Resulting Assets**
  All selected assets will be stored or stored and submitted depending on which action the user chooses on the Import Assets page. The Import Results page will appear indicating the status of the assets involved. In most cases, a large majority of the assets will be in a pending state (influenced by the number of assets as well as the concurrentThread and loadFactor properties). The user will receive an email when all assets have been processed, assuming email-results is true. The email will contain a log file indicating any errors or warnings that might have been encountered.

XMLImporter

Behavior:

Imports assets based on an XML file and a XSLT stylesheet. The importer first transforms the XML file using the XSLT stylesheet. The stylesheet transformation must transform the XML file into another XML file corresponding to a set of assets corresponding to the asset.xsd schema in the library's document source (see example below). The stylesheet will correspond to the type of XML files being processed. The asset type of the assets specified in the resulting XSLT transformed XML file should correspond with an asset prototype mapping definition in the LPC (see example below). The prototype used depends on the name of the importer and asset type of the asset being

[47] For relationships, the value is the combination of the related asset name and version. For by-reference and by-description artifacts, the value is the reference (i.e. URL or the description text).

created. This prototype defines default attributes (version, description, overview, etc.) that are used in the absence of one specified. The template and asset-type used to create the asset always comes from the prototype.

The overview artifact, if specified, should be specified by-description, which will be converted internally to a by-value artifact.

The importer file specified that the XSLT runs against can be populated into the generated assets with either a by-value or by-reference containment. by-description is not supported for the importer file. To signify that the artifact should contain the contents or reference to the file used during import, the artifact reference should be "importer-file". If the file used during the import is specified using a URL, and the containment is by-reference, the generated asset should contain the same URL. Any other scenario (by-value containment, or by-reference containment with an importer file that is uploaded) will result in the file being stored on the asset as a by-value artifact.

- **Properties:**
  - *xslt-document*
    A document in the document source that transforms the XML file into another XML file containing the assets to create. (required)
  - *description*
    The description of the importer.
    Defaults to "Import XML documents". (optional)
- **Parameters:**
  - *Import File*
    The file to be imported. (required)
- **Properties passed into the XSLT transform:**
  The XSLT transform can make use of properties in two ways. The properties specified on the LPC Importer definition are passed to the transform as well as a set of prototype properties based on mapped prototypes for the importer. If the prototype mapping for the importer has been defined (as in the following example) it will pass in properties corresponding to the prototype in the following format:
  - prototype.*key*.name
  - prototype.*key*.version
  - prototype.*key*.type
  - prototype.*key*.template
  - prototype.*key*.description
  - prototype.*key*.overview

- **Resulting Assets**
  Assets may fail creation for a number of reasons. If the asset already exists by name and version in the library, we don't create another. If there are template/type metadata issues in the generated assets, they may not be created. If there are other less serious metadata issues in the assets, they may be created, but fail submission. If an asset has validation errors it will not be submitted for publish.

- **XSLT transformed example**
  This assumes the XSLT transform provided takes in an XML file and produces the following output. It creates two assets, one with a relationship to the other.

  Importer Definition

```
<importer name="Sample Importer" class="XMLImporter">
  <properties>
    <property name="xslt-document" value="xmlimporter-sample.xsl" />
  </properties>
</importer>
```

  LPC Prototype Definitions

```
<asset-type-definition description="Auto-created from asset definition template"
name="Application">
  <templates>
    <template name="Application - General" />
  </templates>
  <edit-roles />
  <prototypes>
      <prototype name = "Default Importer Application">
          <external-mappings>
              <external-mapping owner = "Sample Importer" key =
"DefaultImporterApplication"/>
          </external-mappings>
          <template>Application - General</template>
          <default-version>1.0</default-version>
          <default-description>Default Application Description</default-description>
          <default-overview>Application produced by XMLImporter</default-overview>
      </prototype>
  </prototypes>
</asset-type-definition>
```

  Stylesheet output XML file

```
<?xml version="1.0" encoding="UTF-8"?>
<importer:assets
xmlns:importer="http://www.soa.com/rm/xmlimporter"
xmlns="http://www.soa.com/rm/asset"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.soa.com/rm/xmlimporter xmlimporter-xslt-
output.xsd">
  <asset>
    <name>Sample Asset 1</name>
    <classifiers>
      <classifier>
        <name>asset-type</name>
        <values>
```

```
        <value>DefaultImporterApplication</value>
       </values>
      </classifier>
     </classifiers>
     <artifacts>
      <artifact>
       <name>overview</name>
       <category>overview</category>
       <containment>by-description</containment>
       <reference>This is a sample overview</reference>
      </artifact>
     <artifacts>
      <artifact>
       <name>Application File</name>
       <category>application-file </category>
       <containment>by-value</containment>
       <reference>importer-file</reference>
      </artifact>
     </artifacts>
    </asset>
    <asset>
     <name>Sample Asset 2</name>
     <classifiers>
      <classifier>
       <name>asset-type</name>
       <values>
        <value>DefaultImporterApplication</value>
       </values>
      </classifier>
     </classifiers>
     <related-assets>
      <related-asset>
       <name>predecessor</name>
       <related-asset-name>Sample Asset 1</related-asset-name>
       <related-asset-version/>
      </related-asset>
     </related-assets>
    </asset>
   </importer:assets>
```

Appendix K: Context Replacement Parameters

This appendix describes event-context based replacement parameters that are available within certain listener properties.

Recipient Parameters

These parameters may be used in comma-separated fashion in the "recipient" properties of listeners such as the Message listener:

Group Context

- **group.roleplayers.<role>**
  Roleplayers of the event's context group.

Common Asset

- **asset.submitting-user**
  Submitting user of the context asset.
- **asset.locking-user**
  Current locking user of the context asset.
- **asset.group.roleplayers.<role>**
  Roleplayers of the owning group of the context asset.

Published Asset

- **published_asset.submitting-user**
  Submitting user of the current published version of the context asset.
- **published_asset.subscribers**
  Subscribers of the published version of the context asset.
- **published_asset.related-subscribers.<related asset category>**
  Subscribers of related assets of the context asset based on the specified relationship category.
- **published_asset.contacts.<role>**
  Published asset contact based on the specified contact role.

Library

- **library.roleplayers.<role>**
  Library scoped roleplayers of the specified role e.g. "ACE".
- **library.feedback**
  Library feedback contact.

Event

- **event.user**
  Context user for the event.

Request

- **request.submitting-user**
  Submitting user of context Asset Request

System

- **system.date**
  The current date in the format "yyyy-MM-dd"

Standard Context Replacement Parameters

These parameters are generally used within a Listener's properties and are bracketed with the {} delimiters.

Catalog Asset

- **catalog_asset.id**
  Resolves to the asset Id.

- **catalog_asset.name**
  Resolves to the name of the catalog asset
- **catalog_asset.version**
  Resolves to the version of the catalog asset
- **catalog_asset.url**
  Resolves to the URL to the detail page of the catalog asset.
- **catalog_asset.property.<property name>**
  Resolves to the value of the specified property of the catalog asset.
- **catalog_asset.classifier.<classifier name>**
  Resolves to the value of the specified classifier of the catalog asset.
- **catalog_asset.library_asset.<published asset parameter>**
  Access a parameter from the associated published asset. See Published Asset parameters for values for <published asset parameter>.
- **catalog_asset.related_library_asset.<relationship name>.<published asset parameter>**
  Access a parameter from a related published asset. See Published Asset parameters for values for <published asset parameter>. Note that this will access the only first related asset with the specified <relationship name>.
- **catalog_asset.related_catalog_asset.<relationship name>.<catalog asset parameter>**
  Access a parameter from a related catalog asset. See Catalog Asset parameters for values for <catalog asset parameter>. Note that this will access the only first related asset with the specified <relationship name>.
- **catalog_asset.relationship_context.<relationship name>.<context parameter>**
  Access a parameter from a relationship context. See Relationship Context parameters for values for <context parameter>. Note that this will access the only first related asset context with the specified <relationship name>.

Published Asset

- **library_asset.id**
  Resolves to the id of the published asset.
- **library_asset.urn**
  Resolves to the URN of the published asset.
- **library_asset.url**
  Resolves to the detail page of the published asset.
- **library_asset.name**
  Resolves to the name of the published asset
- **library_asset.version**
  Resolves to the version of the published asset
- **library_asset.property.<property name>**
  Resolves to the value of the specified property of the published asset.
- **library_asset.classifier.<classifier name>**
  Resolves to the value of the specified classifier of the published asset.
- **catalog_asset.catalog_asset.<catalog asset parameter>**
  Access a parameter from the associated catalog asset. See Catalog Asset parameters for values for <catalog asset parameter>.

- **library_asset.related_library_asset.<relationship name>.<published asset parameter>**
  Access a parameter from a related published asset. See Published Asset parameters for values for <published asset parameter>. Note that this will access the only first related asset with the specified <relationship name>.
- **library_asset.related_catalog_asset.<relationship name>.<catalog asset parameter>**
  Access a parameter from a related catalog asset. See Catalog Asset parameters for values for <catalog asset parameter>. Note that this will access the only first related asset with the specified <relationship name>.
- **library_asset.relationship_context.<relationship name>.<context parameter>**
  Access a parameter from a relationship context. See Relationship Context parameters for values for <context parameter>. Note that this will access the only first related asset context with the specified <relationship name>.

Group

Note that the "group" context will be set to the context asset's owning group for catalog events and to the active project for acquisition events.

- **group.id**
  Resolves to the URN of the context group.
- **group.name**
  Resolves to the name of the context group.
- **group.path**
  Resolves to the ancestor path of the context group. The path is represented as an ordered list of ancestor ids separated by spaces starting with the context group itself and leading to the Enterprise group.
- **group.property.<property name>**
  Resolves to the value of the specified property of the context group.

Project

Note that unlike the "group" context, the "project" context will always be set to the active project.

- **project.id**
  Resolves to the URN of the active project.
- **project.name**
  Resolves to the name of the active project.
- **project.path**
  Resolves to the ancestor path of the active project. The path is represented as an ordered list of ancestor ids separated by spaces starting with the project itself and leading to the Enterprise group.
- **project.property.<property name>**
  Resolves to the value of the specified property of the context group.

User

- **user.id**
  Resolves to the URN of the context user.

- **user.account**
  Resolves to the account name of the context user.
- **user.fullname**
  Resolves to the full name of the context user.
- **user.email**
  Resolves to the email of the context user.
- **user.attribute.<attribute name>**
  Resolves to the value of the specified attribute for the context user.
- **user.property.<property name>**
  Resolves to the value of the specified property for the context user.
- **user.metric.<event type>**
  Resolves to the value of the specified event metric for the context user.
- **user.has_project_role.<role name>**
  Resolves to "true" if the user has the specified role for the context (active) project, else resolves to "false".

Library

- **library.id**
  Resolves to the id of the current library.
- **library.name**
  Resolves to the name of the current library.
- **library.base_url**
  Resolves to the base URL of the current library.
- **library.context_root**
  Resolves to the context root of the current library.
- **library.birt_root**
  Resolves to the BIRT root of the current library.
- **library.property.<property name>**
  Resolves to the value of the specified library-scoped property.

Event

- **event.type**
  Resolves to the name of the event.
- **event.time**
  Resolves to the time of the event.
- **event.property.<property name>**
  Resolves to the value of the specified event property.

Request

- **request.id**
  Resolves to the id of the context Asset Request.
- **request.state**
  Resolves to the current state of the context Asset Request.
- **request.type**
  Resolves to the type of the Asset Request.
- **request.url**
  Resolves to the detail page of the Asset Request.
- **request.property.<property name>**
  Resolves to the value of the specified property on the context Asset Request.
- **request.library_asset.<published asset parameter>**
  Access a parameter from the request's associated published asset. See Published Asset parameters for values for <published asset parameter>.
- **request.catalog_asset.<catalog asset parameter>**
  Access a parameter from the request's associated catalog asset. See Catalog Asset parameters for values for <catalog asset parameter>.
- **request.consuming_asset.<catalog asset parameter>**
  Access a parameter from the request's associated consuming catalog asset. See Catalog Asset parameters for values for <catalog asset parameter>. Note that this is only applicable for an asset-based acquisition request.

- **request.consuming_relationship**
  Resolves to the name of the consuming relationship for an asset-based acquisition request.

Relationship Context

Note that Relationship Context parameters are only accessible indirectly through catalog and published asset relationships (see Catalog and Published Asset parameters above).

- **relationship_context.id**
  Resolves to the id of the Relationship Context.
- **relationship_context.state**
  Resolves to the current state of the Relationship Context.
- **relationship_context.url**
  Resolves to the URL of the Relationship Context.
- **relationship_context.source_asset.<catalog asset parameter>**
  Access a parameter from the Relationship Context's source (consuming) catalog asset. See Catalog Asset parameters for values for <catalog asset parameter>.
- **relationship_context.target_asset.<published asset parameter>**
  Access a parameter from the Relationship Context's source (consuming) published asset. See Published Asset parameters for values for <published asset parameter>.
- **relationship_context.relationship_name**
  Resolves to the name of the relationship this Relationship Context is associated with.
- **relationship_context.property.<property name>**
  Resolves to the value of the specified property on this Relationship Context .

Appendix L: Artifact-Comparators

The following artifact-comparator classes are currently available:

SampleXMLArtifactComparator

- **Class:** com.soa.repositorymgr.extensions.SampleXMLArtifactComparator
- **Intended Target Artifacts**
  - *XML*
  - *WSDL*
- **Behavior:**
  Uses an XML difference algorithm coupled with optional "pre" and "post" XSL transforms on input documents and result document respectively. Can be used in a "generic" mode for standard XML differences or with specific transforms packaged with Lifecycle Manager for WSDL specific transforms. Note that the SampleXMLArtifactComparator source is provided in the extensions jar and may be customized to meet specific needs, including the insertion of a custom difference engine.
- **Properties:**
  - *output-style*
    This property determines the output form of the internal difference engine. Choices are:
    in-source-comments – indicates that XML difference comments will be inserted into a copy of the source document

in-source-elements – indicates that XML wrapper elements will be inserted into a copy of the source document
external-document – Indicates that changes will be shown as XML elements in a separate document
external-document-truncated – Similar to *external-document* but child elements of changed elements will not be shown in the output document. This property is optional and will default to *external-document-truncated* if not specified.

- *pre-transform-document-id*
  This property is used to specify an XSLT document in Lifecycle Manager's internal document repository to be applied to the artifact contents prior to being processed by the differences engine. Such a transform may be used to simplify or organize an XML document in a way that makes the generated differences more consumable. (Optional)
- *post-transform-document-id*
  This property is used to specify an XSLT document in Lifecycle Manager's internal document repository to be applied to the output document of the differences engine. Such a transform may be used to render the XML output into HTML. (Optional)
- *final-content-type*
  This property is used to indicate to the Lifecycle Manager UI the type of the final output document. This is necessary when a post-transform converts the type of the document. This property is optional and if not specified will default to the content-type output by the difference engine: "text/xml".

- **Suggested Configurations**
  For WSDL Artifacts:

```
   <artifact-comparator name="WSDLComparator"
class=
   "com.soa.repositorymgr.extensions.SampleXMLArtifactComparator">
   <properties>
     <property name="pre-transform-document-id"
         value="wsdlpreproc.xsl" />
     <property name="post-transform-document-id"
         value="wsdlpostproc.xsl" />
     <property name="output-style"
         value="external-document-truncated" />
     <property name="final-content-type" value="text/html" />
   </properties>
   <content-type>wsdl</content-type>
</artifact-comparator>
```

  For XML Artifacts:

```
<artifact-comparator name="XMLComparator"
class=
   "com.soa.repositorymgr.extensions.SampleXMLArtifactComparator">
   <properties>
     <property name="output-style"
         value="in-source-comments" />
   </properties>
   <content-type>xml</content-type>
</artifact-comparator>
```

Appendix M: Value Sources

The following Value Source classes are currently available:

ExistingClassifierValues

- **Applicability**
  Non-compound Classifiers
- **Resolution**
  Dynamic
- **Behavior:**
  Used to present a list of classifier values that have already been used for the specified classifier.
- **Properties:**
  - *none*

LDAPUserValueSource

- **Applicability**
  Classifiers/Properties
- **Resolution**
  Dynamic
- **Behavior:**
  Queries LDAP users from the configured LDAP repository and returns the list in a customizable format using attributes from the LDAP records.
- **Properties:**
  - *display-format*
    This property is used to specify the format of the resulting users retrieved from LDAP. The default format is "{Full Name} - {Email}". The substitution parameters, enclosed in curly braces are those attribute definitions that have been mapped to LDAP. These can be retrieved using the GetCurrentAttributeDefinitions and SetAttributeDefinitions administration commands. Optional.
  - *search-field*
    A search field to use when querying LDAP records. If the search field property is specified, the user will be presented with a dialog where they can enter search criteria to limit the number of users returned. The search field must be set to an existing mapped user attribute (e.g. "Last Name", see the display-format property). Optional

OWLValueSource

- **Applicability**
  Classifiers/Properties
- **Resolution**
  Static
- **Behavior:**
  Constructs values from an ontology document in OWL format. Values will be produced

in compound form and will have as many levels as the depth of the target ontology. The OWL document can be referenced by URL or stored locally in the Lifecycle Manager Document Repository.

- **Properties:**
  - *owl-document-url*
    This property is used to specify the URL of the OWL document in the remote reference case. This property is optional but either *owl-document-url* or *owl-document-id* must be provided.
  - *owl-document-id*
    This property is used to specify the file name of an OWL document in the Lifecycle Manager Document Repository. This property is optional but either *owl-document-url* or *owl-document-id* must be provided.
  - *owl-root-class*
    This property is used to specify the class within the OWL document that is to be used as the root of the value tree. (Mandatory)
  - *mapping-key*
    The OWLValueSource will associate external mappings with the values it produces. These may be useful in integrating Lifecycle Manager with other systems of record. This property allows the mapping to be specified. The property is optional and a default mapping key of "OWL" will used if not specified.

RemoteGroupValueSource

- **Applicability**
  Classifiers, Request Properties
- **Resolution**
  Dynamic
- **Behavior:**
  Creates a list of groups and/or projects from a remote Policy Manager or Lifecycle Manager system. The list can include both projects and groups, or only groups or only projects.
- **Properties:**
  - *federated-system-name*
    This name of the federated system corresponding to the remote Portfolio Manager or Lifecycle Manager system (required).
  - *include-groups*
    Determines if the list contains groups The default is "true" (optional).
  - *include-projects*
    Determines if the list contains projects. The default is "true" (optional).
  - *dynamic-resolution*
    Determines if the remote groups are queried each time they need to be enumerated. If set to false, the results are cached the first time they are queried, otherwise the remote system is queried every time. The default is "true" (optional).

RolePlayerValueSource

- **Applicability**
  Request Properties
- **Behavior:**
  Creates a list of user-ids of role-players for the specified group-role and owning Group of the target Asset Request. The initial role-player in this list is designated as a default value for the property.
- **Properties:**
  - *role*
    This property is used to specify the Group role used in selecting the role-player list (required).
  - *format*
    This property determines how role-player values will be displayed. Choices are:
    FULL_NAME (default) – Uses the full name format from LDAP for the user. This is commonly "<first name> <last name>".
    LAST_FIRST – Indicates that names are in the form "<last name>, <first name>"
  - *ordering*
    This property determines the sort order for the values. Choices are:
    HIERARCHY (default) - Ordered with role-players at the lowest levels of the Group structure first.
    ALPHA – Order alphabetically across all role-players by formatted name.

ScriptPropertyValueSource

- **Applicability**
  Request Properties
- **Behavior:**
  ScriptPropertyValueSource is a generic script based value-source. It can be used either as an interactive or non-interactive value-source for AssetRequest properties.
- **Properties:**
  - *script-id*
    This is the name of the script to invoke. This script must be uploaded using the StoreDocument command or Configuration Designer. See the System Administration guide for more information on the StoreDocument command. (Mandatory).
  - *script-type*
    This determines which interpreter to execute the script. Valid values are "jython" or "beanshell". If this parameter is not specified, the interpreter used depends on the script-id's extension (.py or .bsh). (Optional)
  - *input-field-name*
    This property specifies the name of the input field used in interactive mode. Note that specification of this property causes the ScriptPropertyValueSource to operate as an interactive value source. (Optional).
  - *input-field-type*
    Specifies the data type of the input field (when *input-field-name* is specified). Accepted values are: "string", "date", "decimal" and "boolean". Default value is "string". (Optional)
- **Script Details**
  - *Non-interactive Mode (input-field-name property not specified)*
    Parameters:

Name	*Class*	*Type*
libraryId	String	IN
propertyName	String	IN
results	Java.util.List	INOUT

    Script implementation is responsible for placing com.logiclibrary.external.PropertyValue objects in the "results" List.
  - *Interactive Mode (input-field-name property specified)*
    Parameters:

Name	*Class*	*Type*
libraryId	String	IN
propertyName	String	IN
filterFields	com.soa.repositorymanager.external.SearchField[]	IN
Context	com.soa.repositorymanager.external.Context	IN
propertyContainer	com.logiclibrary.external.CustomPropertyContainer	IN
Results	com.soa.repositorymanager.external.PropertyValueResults	INOUT

    Script implementation is responsible for placing com.logiclibrary.external.PropertyValue objects in PropertyValueResults object "results".

SQLValueSource

- **Applicability**
  Classifiers or Request Properties
- **Resolution**
  Static or Dynamic
- **Behavior:**
  Retrieves data from either the Repository or Portfolio manager database or an external database. The SQL can contain bind parameters which can be sourced from either the user or an asset.
- **Properties:**
  - *sql*
    The SQL that is executed. The resulting query should return two columns named "vs_key" and "vs_value". It can optionally also return columns named "vs_description" and "vs_url" in the case of request properties. The key is used to uniquely identify the row returned and is used internally. The value is what is displayed to the user. The description and url values for request properties are additional information displayed when acquiring assets. (required)
  - *bind-param-x*
    Bind parameters that can be substituted into the SQL to dynamically modify the results. If the sql property is "select * from tablename where tablecolumn is ?", then a property bind-param-1 should be specified. This property can be a fixed string, or it can be a replacement parameter such as "{catalog_asset.version}" or a a search field parameter (e.g. "{search-field-1}") (optional). For more information on replacement parameters see this appendix.
  - *search-param-x*
    A list of properties (e.g. "search-param-1", "search-param-2"), that are displayed to the user if "active-resolution" property is set to true. These search parameters can be used to modify the results of the SQL through the bind-param properties. "search-param-1" is supported currently for classifiers only (optional). For more information on replacement parameters see this appendix.
  - *url*
    The JDBC url of the external database to run the SQL against. If the url is not specified, the internal Portfolio or Lifecycle Manager JNDI datasource will be used. If url is specified, then the driver, user, and password properties are required. (optional).
  - *driver*
    The JDBC driver class to use to connect to the database. Required if the *url* property is specified.
  - *user*
    The user used to connect to the remote database (optional). Required if the *url* property is specified.
  - *password*
    This user's password used to connect to the remote database. Required if the *url* property is specified.
  - *dynamic-resolution*
    This property determines whether the results of execution are cached after the first

execution. This property should not be set to false if the active-resolution property is set to true, or if any bind parameters are dynamic. The default is "true" (optional)

- *active-resolution*
  This property determines if the user is displayed the search parameters when the SQL is executed (required).

- **Example Entry:**

```
 <value-source name="sql-value-source" class="SQLValueSource">
   <properties>
     <property name="sql" value="select libname as vs_value, libraryid as
vs_key from foundation_dbinfo where libname like ? order by libname"/>
     <property name="bind-param-1" value="{search-field-1}"/>
     <property name="search-field-1" value="Library Name"/>
     <property name="url"
value="jdbc:oracle:thin:@oracle10.example.com:1521:DATABASE"/>
     <property name="driver" value="oracle.jdbc.OracleDriver"/>
     <property name="user" value="user"/>
     <property name="password" value="password" encrypt="true"/>
     <property name="dynamic-resolution" value="true"/>
     <property name="active-resolution" value="true"/>
   </properties>
 </value-source>
```

WSDLElementValueSource

- **Applicability**
  Request Properties
- **Resolution**
  Dynamic
- **Behavior:**
  Creates a list of values based on the contents of WSDL element names (services, ports, bindings, port types, or operations). The value source is configured for only one element type – it cannot be configured to display both operations and services. The value source will attempt to build up the entire WSDL (using imported WSDLs). Therefore, configuration properties can be specified in much the same way as other WSDL validators and listeners.
- **Properties:**
  - *wsdl-element-type*
    This property is used to specify the WSDL element types that will be enumerated. Valid values are: "service", "port", "binding", "portType", "operation". (Mandatory)
  - *operation-format*
    This property determines how operations will be displayed. If *operation-format* is set to "name", only the operation's name will be displayed. If "interface" is specified, the format will resemble *portTypeName.operationName*. If "full" is specified, the format will resemble *portTypeName.operationName([in] message,...)*. Valid values: "name", "interface", "full". Default: "name"

- *consuming-asset*
  This property determines if operations from the consumed or consuming asset are displayed. If *consuming-asset* is set to "false", the operations from the consumed asset will be displayed. If *consuming-asset* is set to "true" then operations from the consuming asset will be displayed. Valid values: "false", "true". Default: "false"
- *display-nothing-found-message*
  Determines if a message (via an actual null value) is displayed. If this property is set to *true* and a WSDL is not specified on the asset, a WSDL cannot be constructed, or there are no WSDL elements for the configured type, a corresponding message will be displayed, otherwise nothing will appear. Valid values: "true", "false". Default: "false"
- *display-qname*
  Determines if various type names are qualified by their containing namespace. This doesn't apply to operations configured to display only their name (operation-format=name). For example, service "Foo" might appear as "{http://example.com/service/foo}Foo". This property is only meant to be used in scenarios with name collisions. Valid values: "true", "false". Default: "false"
- *service-artifact-category*
  This is the artifact category containing the service (WSDL). The default is "message-definition"
- *packed-service-artifact-category*
  This is the artifact category containing a packed service (ZIP). The service and relevant schemas can be packaged together in this artifact. The default is "packed-service"
- *service-namespace-classifier*
  The namespace classifier that contains the target namespace of the service. This is used to resolve services that are located within Lifecycle Manager. The default is "target-namespace".
- *service-name-classifier*
  The name of the classifier that contains the service name that the asset represents. Multiple services within a WSDL will cause errors if this classifier is not set. The default is "service-name".
- *schema-artifact-category*
  This is the artifact category containing schemas. The default is "schema-definition"
- *schema-namespace-classifier*
  The namespace classifier that contains the target namespace of the schema. This is used to resolve schemas that are located within Lifecycle Manager. The default is "target-namespace".

Appendix N: Federated Systems

The following Federated System classes are currently available:

FederatedRepository

The FederatedRepository class is used to represent a remote Lifecycle Manager or Portfolio Manager library.

- **Used By:**
  - RemoteAssetPublisher listener.
- **Properties:**
  - *Remote-host*
    URL of remote installation including the application context name. For example: *http://myserver.com/rm* (Mandatory)
  - *remote-library-name*
    Name of the remote library. (Mandatory)
  - *user*
    User Id used in connecting to the remote library
  - *password*
    user password used in connectiong to the remote library

ClearCaseSystem

The ClearCaseSystem defines the basic information regarding a ClearCase client. A ClearCaseSystem is required by the PublishArtifactToClearCase listener and the ClearCaseArtifactSource.

- **Used By:**
  - PublishArtifactToClearCase listener.
  - ClearCaseArtifactSource
- **Properties:**
  - *description*
    An optional description for the system
  - *cleartool*
    (Required) Path to the Rational ClearCase cleartool executable.
  - *ucm*
    (Required) True or false, indicating if the system represents a Base ClearCase installation or UCM.
  - *view-root*
    (Required) ClearCase location where views are accessible. If view_type=dynamic, this should be where the dynamic views are mounted. If view_type=snapshot, this should be where the snapshot views are stored.
  - *view-store*
    (Required) ClearCase view store location.
  - *view-prefix*
    A prefix used to compose the view used to house the artifact. If not provided, a default of "`lgdxvw_`" will be used. The resulting view name will take the form "`<view-prefix><hostname><libID>`" and a suffix of "`_<baselineName>`" will be added if `ucm=true`. Note that `<hostname>` will be the name of the local host. This ensures every node in a cluster will generate a unique view name.
  - *view-name*

The name of the view used to house the artifact. Specifying this property will avoid construction of the view name from the view-prefix property above. NOTE: The property must not be specified if running in a clustered environment as it would cause all nodes in a cluster to have the same view name.

- *view-type*
  (Required) ClearCase view type to create when accessing file elements, either "dynamic" or "snapshot".
- *view-refresh*
  (Applicable if view-type=snapshot) Indicates if Lifecycle Manager should refresh the snapshot when accessing file elements. Default = false.
- *vob-prefix*
  The vob prefix (typically only necessary for on non-Windows machines). It is the path to prepend to a VOB reference on UNIX. For example, suppose you have a VOB on Windows \sample_vob, and this same VOB is used on UNIX as /vobs/sample_vob. The VOB prefix should be set to "/vobs".
- *pvob*
  (Required) The pvob (UCM) or vob (Base ClearCase) used to store the artifact
- *artifact-source-name*
  (Required) The name of the artifact-source as specified in the artifact-source definition.

- **Example LPC Entry:**

```
<federated-system name="ClearCaseSystem" class="ClearCaseSystem">
  <properties>
    <property name="cleartool" value="/opt/rational/clearcase/bin/cleartool" />
    <property name="ucm" value="false" />
    <property name="view-root" value="/view/" />
    <property name="view-store" value="/home/repoman/.view_store" />
    <property name="view-name" value="the_dynamic_view" />
    <property name="view-type" value="dynamic" />
    <property name="vob-prefix" value="/vobs" />
    <property name="pvob" value="/the_vob" />
    <property name="artifact-source-name" value="cc" />
  </properties>
</federated-system>
```

WebDAVSystem

The WebDAV defines some basic information needed to connect to a WebDAV server.

- **Used By:**
  - PublishArtifactToWebDAV listener.
- **Properties:**
  - *description*
    An optional description for the system
  - *url*
    (Required) The host of the WebDAV server. This should not contain any path information (specified in PublishArtifactToWebDAV). Example: http://subversion:8080
  - *user*
    (Required) The WebDAV user's authentication credentials. This user will be used for operations requiring authentication (to check-in files).
  - *password*

(Required) The WebDAV user's password.

- **Example LPC Entry:**

```
<federated-system name="WebDAVSystem" class="WebDAVSystem">
  <properties>
    <property name="url" value="http://subversion:8080" />
    <property name="user" value="johndoe" />
    <property name="password" value="password" />
  </properties>
</federated-system>
```

Appendix O: Configuration Document Repository

Certain extension classes such as Listeners and Artifact Comparators may require access to configuration documents such as XSL transforms or scripts. For this purpose, the Lifecycle Manager and Portfolio Manager products provide a repository for such documents. Documents in the repository may be text or binary and are identified with a unique Document ID. Often the file name of a document will serve as its Document ID. SOA Software pre-defines a number of documents in the repository required by internally provided extension classes such as SampleXMLArtifactComparator. These documents are referred to as "global" and may be overlaid by a local but not removed. Additional ("local") documents can be added to the repository and maintained through a set of administrative commands:

StoreDocument

This command stores a document in the repository. It will replace an existing local document with the same Document ID or will overlay a global document with the same Document ID.

Parameters:

- *Library* must be set to appropriate library
- *File Parameter* must be set with the document to store
- *Parameter 1* must be set to the Document ID. If Parameter 1 is not specified the filename will be used as the Document ID.

RemoveDocument

This command removes a document from the repository. Note that a global document cannot be removed.

Parameters:

- *Library* must be set to appropriate library
- *Parameter 1* must be set to the Document ID of the document to remove

GetDocument

Retrieve a document from the repository. Both local and global documents may be accessed with this command. In the case where a local document overlays a global with the same Document ID, the local document will be returned.

Parameters:

- *Library* must be set to appropriate library
- *Parameter 1* must be set to the Document ID of the document to retrieve

GetDocumentIDs

Retrieve a list of Document IDs of both local and global documents in the repository.
Parameters:

- *Library* must be set to appropriate library

Appendix P: Extension Programming Environment

The Lifecycle Manager and Portfolio Manager products define a number of extension points described earlier in this document that use a simple factory pattern utilizing an SOA Software provided abstract base class. In most cases a number of pre-defined subclasses are provided by SOA Software for common behavior, some are highly customizable through their configuration properties. However, it may be necessary at times for a customer to provide their own subclasses to define required custom behavior.

For this reason the Lifecycle Manager and Portfolio Manager products have defined a local extension programming environment to allow custom extension point classes to be compiled and run in. This environment can be found in the "extension-apis.jar" provided within the Lifecycle Manager or Portfolio Manager EAR file.

API Package

The "com.soa.repositorymgr.extension.apis" package contains base classes and helper classes that make up the programming environment. In particular, the class ExtensionHelper.java provides a set of useful helper methods as well as methods to allow access to the Automation and external API packages[48]. Javadoc is provided for this package. Base classes are provided that correspond to extension points described elsewhere in this document. Often these base classes will provide methods useful in implementing subclasses. The current set of base classes are:

- *Listener*
- *ArtifactComparator*.
- *ArtifactSource*
- *ArtifactTransform*
- *Importer*
- *ValueSource*
- *AssetValidator*

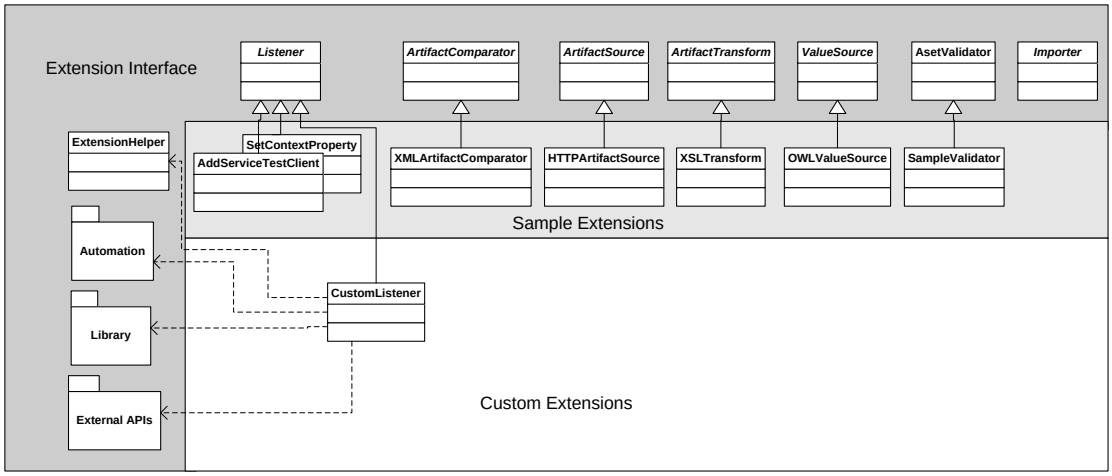
Extensions Package

The "com.soa.repositorymgr.extensions" package contains sample extension classes provided by SOA Software. Source code and Javadoc is available for classes in the "com.soa.repositorymgr.extensions" package. These classes are meant to service as starting points for customer provided classes.

Diagram

The following diagram shows the class structure of the extensions environment:

[48] These are essentially the same classes available to remote clients. For additional information contact SOA Software support.



Building and Deploying

Customer classes should be compiled against the set of jars packaged in the Automation Extensions download from the Library Download Center. Deployment of custom extension classes is described in the Library Administration Guide.

Appendix Q: Validators

Asset Validators

The following Asset Validator classes are currently available:

AssociateAssetsByClassifier

- **Class:** `com.logiclibrary.validators.AssociateAssetsByClassifier`
- **Behavior:**
  The AssociateAssetsByClassifier validator looks up a configured set of assets based on classifier value and associates any assets with that name and version to the asset being validated. If there are no such assets found, an unresolved relationship is created to the mapped asset. The classifier value to asset mappings are configured in a mapping document stored in the document source. If a *mapping-classifier* classifier exists on the asset, all assets matching the classifier values in the *mapping-document-id* document will be added to the asset. Multiple asset relationships are created if a classifier value exists more than once. If there is more than one asset with the same name and version relationships to all matching assets will be created.
- **Properties:**
  - *mapping-document-id*
    This is the document source id of the document containing the mappings. The format of the file is standard CSV format containing <classifiervalue,assetname,assetversion> tuples. See below for an example. Required.
  - *mapping-classifier*
    The classifier that contains the values that will be cross-referenced with the

mapping document to determine which asset relationships will be added to the asset. Required.

- *mapping-relationship*
  The relationship type that will be used for any relationships that are added to other assets. Required.

- **Example Mapping Document:**
  B2B,Business Interaction Service,1.0
  Insurance,Insurance Service,1.0

ConditionalValidator

- **Class:** `com.logiclibrary.registry.ConditionalValidator`
- **Behavior:**
  Supports cross-element validation capabilities during asset edit (i.e., synchronous validation).

- **Properties:**
  - *Assertions-document-id*
    Name of the XML document specifying the assertions to be validated by an instance of this class. This document must be stored in the Configuration Document Repository of the library for which the class is configured (see Appendix O for details) and must conform to the schema assetvalidation.xsd, also available for retrieval from the Configuration Document Repository. (Mandatory)
- **Note:**
  For performance reasons, the ConditionalValidator class caches the contents of the designated assertions XML document. Because of this, it is necessary to reload the Library Configuration Document to allow the ConditionalValidator to pick up changes after the assertion document is modified.

The Conditional Validator class supports the concept of validation assertions. A validation assertion combines zero or more optional condition-filters specifying the entry condition of this assertion with one or more expected-filters specifying the asset contents expected by this assertion.[49] In the event that the asset contents are not as expected, the assertion also specifies a validation-message that indicates the severity (INFORMATIONAL, WARNING or SEVERE), affected asset metadata element id (as specified in the global definition template) and type (ASSET, IDENTIFIER, CLASSIFIER, ARTIFACT or RELATED_ASSET) and validation message to be displayed to the end user.

Condition-filter and expected-filter elements are of type asset-filter. Asset-filter instances are composed any combination of identifier-filter, classifier-filter, artifact-filter and related-asset-filter elements (i.e., elemental filters). These elements are logically ANDed together to establish entry criteria (in the case of condition-filter) or the expected result set (in the case of expected-filter).

Following are some elemental filter examples:

Checks if asset version is set to the value “1.0”:

```
<p:identifier-filter name="V1.0">
   <p:identifier-criteria identifier-id="VERSION">
      <p:value-set>
         <p:value value="1.0" />
      </p:value-set>
   </p:identifier-criteria>
</p:identifier-filter>
```

Uses Perl 5 regular expression to check if asset version is formatted as V<one or more digits>.<one or more digits> (see http://search.cpan.org/dist/perl/pod/perlre.pod for a description of expression syntax):

```
<p:identifier-filter name="Version Format">
   <p:identifier-criteria identifier-id="VERSION">
      <p:value-set>
```

[49] If no condition-filters are specified for an assertion, the assertion is always applied to any asset being validated. If multiple condition or expected filters are specified, they are logically ORed together to form the entrance criteria or expected asset content for this assertion.

```
         <p:value value="V\d*\.\d*" regular-expression="true" />
      </p:value-set>
   </p:identifier-criteria>
</p:identifier-filter>
```

Checks if the asset-type classifier is set to the value “Service”:

```
<p:classifier-filter name="Type:Service">
   <p:classifier-criteria classifier-name="asset-type">
      <p:value-set>
         <p:value value="Service" />
      </p:value-set>
   </p:classifier-criteria>
</p:classifier-filter>
```

Checks if the protocol classifier is set to the value “http”:

```
<p:classifier-filter name="Protocol:http">
   <p:classifier-criteria classifier-name="protocol">
      <p:value-set>
         <p:value value="http" />
      </p:value-set>
   </p:classifier-criteria>
</p:classifier-filter>
```

Checks if a documentation artifact exists:

```
<p:artifact-filter name="Documentation Exists">
   <p:artifact-criteria artifact-category="documentation"
      exists="true">
   </p:artifact-criteria>
</p:artifact-filter>
```

Checks if a uses-schema relationship exists and resolves to a live asset:

```
<p:related-asset-filter name="uses-schema resolved">
   <p:related-asset-criteria related-asset-name="uses-schema"
      resolved="true">
   </p:related-asset-criteria>
</p:related-asset-filter>
```

And following are examples showing how these elemental filter instances can be combined to create Asset Filters:

```
<p:asset-filter name="V1.0 http-based Services">
   <p:identifier-filter name="V1.0" />
   <p:classifier-filter name="Type:Service" />
   <p:classifier-filter name="Protocol:http" />
</p:asset-filter>
<p:asset-filter name="Documentation Exists">
   <p:artifact-filter name="Documentation Exists" />
</p:asset-filter>
<p:asset-filter name="uses-schema resolved">
   <p:related-asset-filter name="uses-schema resolved" />
</p:asset-filter>
<p:asset-filter name="Version Format">
   <p:identifier-filter name="Version Format" />
</p:asset-filter>
```

And finally, some examples of assertions using the above Asset Filters:

Raises an informational message that asset version is not of the form V<at least one digit>.<at least one digit>:

```
<p:assertion name="Asset version not of the form Vx.y">
   <p:expected-filter>Version Format</p:expected-filter>
   <p:message result-type="INFORMATIONAL" element-id="VERSION"
      element-type="IDENTIFIER">
      <p:message>
         <p:default-message> Asset version is not of the form Vx.y where x
and y are zero or positive integers</p:default-message>
      </p:message>
   </p:message>
</p:assertion>
```

Raises a warning message that documentation is missing from a V1.0 http-based Service asset:

```
<p:assertion name="V1.0 http-based Services should have Documentation">
   <p:condition-filter>V1.0 http-based Services</p:condition-filter>
   <p:expected-filter>Documentation Exists</p:expected-filter>
   <p:message result-type="WARNING" element-id="documentation"
      element-type="ARTIFACT">
      <p:message>
         <p:default-message>V1.0 http-based Services should have
Documentation</p:default-message>
      </p:message>
   </p:message>
</p:assertion>
```

Enforces that Service assets must have at least one resolved uses-schema relationship:

```
<p:assertion
   name="Services Assets must have uses-schema relationship">
   <p:condition-filter>Type:Service</p:condition-filter>
   <p:expected-filter>uses-schema resolved</p:expected-filter>
   <p:message result-type="SEVERE"
      element-id="uses-schema" element-type="RELATED_ASSET">
      <p:message>
         <p:default-message>Service Assets must have at least one live uses-
schema relationship</p:default-message>
      </p:message>
   </p:message>
</p:assertion>
```

The above elements are placed in the validation document as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<p:asset-validation xmlns:p="http://www.soa.com/rm/assetvalidation"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.soa.com/rm/assetvalidation
assetvalidation.xsd ">
   <p:asset-filters>
      <!-- asset filter instances go here →
   </p:asset-filters>
   <p:identifier-filters>
      <!-- identifier filter instances go here →
   </p:identifier-filters>
```

```
   <p:classifier-filters>
      <!-- classifier filter instances go here →
   </p:classifier-filters>
   <p:artifact-filters>
      <!-- artifact filter instances go here →
   </p:artifact-filters>
   <p:related-asset-filters>
      <!-- related asset filter instances go here →
   </p:related-asset-filters>
   <p:assertions>
      <!-- assertion instances go here →
   </p:assertions>
</p:asset-validation>
```

RelatedAssetFilterComplianceValidator (Deprecated – see RelatedAssetValidator)

- **Class:** `com.logiclibrary.registry.RelatedAssetFilterComplianceValidator`
- **Behavior:**
  Validates that related assets comply with the filters assigned to their corresponding relationships. Note that validation is not applicable for related-assets not bound to an asset.
- **Properties:**
  - *severity*
    Severity of the validation messages issued for non-compliance. Values are "SEVERE", "WARNING", and "INFORMATIONAL". This property is optional and defaults to "SEVERE".

RelatedAssetValidator

- **Class:** `com.logiclibrary.registry.RelatedAssetValidator`
- **Behavior:**
  Validates the following:
  - Related assets (as published in the library) are in compliance with any filter or search constraining the relationship from this asset
  - This asset complies with any filter or search constraining relationships from published assets to this asset.
  - Existence of acquisition relationships that are pending initial submission
  - Existence of acquisition relationships without a completed acquisition (including those with a revoked acquisition)

  Each of these validations is controlled by an associated severity property as described below.
- **Properties:**
  - *related-asset-non-compliant-severity*
    This property controls whether to display messages about non-compliant related-assets. This property can take the values of "off", " informational", "warning", or "severe". The default is "warning".
  - *asset-non-compliant-severity*
    This property controls whether to display messages about this asset being non-compliant with references from other published assets. This property can take the

values of "off", " informational", "warning", or "severe". The default is "warning".

- *related-asset-pending-submission-severity*
  This property controls whether to display messages about acquisition relationships that are not yet submitted. This property can take the values of "off", " informational", "warning", or "severe". The default is "warning".
- *related-asset-acquisition-not-completed-severity*
  This property controls whether to display messages about acquisition relationships without a completed acquisition (including those with a revoked acquisition). This property can take the values of "off", "informational", "warning", or "severe". The default is " informational".

SchemaValidator

- **Class:** `com.logiclibrary.registry.SchemaValidator`
- **Behavior:**
  Validates various aspects of a schema artifact including whether or not include/import references can be resolved and if the namespace classifier is invalid (if it exists).
- **General Schema Properties:**
  - *schema-artifact-category*
    This is the artifact category containing schemas. The default is "schema-definition"
  - *schema-namespace-classifier*
    The namespace classifier that contains the target namespace of the schema. This is used to resolve schemas that are located within Lifecycle Manager. The default is "target-namespace".
  - *schema-location-classifier*
    The location classifier that contains the location of the schema. This is used to resolve schemas that referenced within Lifecycle Manager. The default is "target-namespace".
- **Properties:**
  - *missing-governed-assets-severity*
    This property controls whether to flag any schemas that are resolved externally. The purpose is to enforce that any used XSDs are resolved by namespace from within Lifecycle Manager. This property can take the values of "off", "info", "warning", or "severe". The default is "off".
  - *unresolved-resources-severity*
    This property controls whether to display warnings about missing imported/included schemas it can't find. This property can take the values of "off", "info", "warning", or "severe". The default is "severe".
  - *populate-classifiers*
    This Boolean property controls whether to add the namespace classifier and location classifier (if the Schema is by reference) to the asset. Valid values are "true" or "false". Default: true.
  - *overwrite-classifiers*
    This Boolean property controls whether to overwrite the namespace classifier and

location classifier (if the Schema is by reference) on the asset. Valid values are "true" or "false". Default: false.

- *create-relationships*
  This Boolean property controls whether to add relationships to resources referenced by the schema. Valid values are "true" or "false". Default: false.
- *referenced-schema-relationship*
  This property defines the relationship type for schemas imported by the asset's schema. Default: related-schema.
- *included-schema-relationship*
  This property defines the relationship type for schemas included by the asset's schema. Default: includes.
- *unpublished-related-asset severity*
  If a related asset isn't published when a relationship is created, this property will define the severity of the unpublished asset message that is displayed to the user. Valid values are "off", "info", "warning", or "severe". Default: severe.

ScriptValidator

- **Class:** `com.logiclibrary.registry.ScriptValidator`
- **Behavior:**
  Supports cross-element validation capabilities during asset edit (i.e., synchronous validation) using a script (beanshell or jython).
- **Properties:**
  - *script-id*
    Name of the script that is executed to validate the asset. This property must be specified. This document must be stored in the Configuration Document Repository of the library for which the class is configured (see Appendix O for details).
  - *script-type*
    Type of the script. This can either be set to "beanshell" or "jython". This parameter is optional, as the type of script will be inferred from the script-id if possible.
- **Variables:**
  There are several variables that are set prior to the script being run. These variables pass in certain information about what is being validated as well as handle validation messages to display to the user.
  - *asset* (com.logiclibrary.automation.asset.Asset) in/out
    This object will usually be the starting point to retrieve classifiers, artifacts, and relationships and generate validation messages based on those entities. The asset object can be modified in the script (e.g. to set the value of a classifier)
  - *context* (com.soa.repositorymanager.external.Context) in
    This object contains context information about the Asset being validated: library, user, etc.
  - *event* (com.logiclibrary.external.notification.Event) in
    This object contains context information about the Asset being validated: library, user, etc.

- *results* (com.soa.logiclibrary.external.ValidationMessages) in/out
  This object is a container that holds the validation messages generated during the script's execution. These messages will be displayed to the end user after successful completion of the script.

The Script Validator class can make use of any APIs exposed through the extensions.zip package.

Here is a snippet of the Process Configuration to configure a ScriptValidator

```
<asset-validators>
   <asset-validator name="ScriptValidator"
      class="com.logiclibrary.registry.ScriptValidator">
      <properties>
         <property name="script-id" value="validation.bsh"/>
      </properties>
      <asset-filter-name>Filter Sync Required</asset-filter-name>
   </asset-validator>
</asset-validators>
```

Here is an example that flags all classifiers with the value of "test" as invalid.

```
import com.soa.repositorymanager.external.ExternalMessage;
import com.soa.repositorymanager.external.ValidationMessage;
import com.soa.repositorymanager.external.ValidationMessages;

classifiers = asset.getAllClassifiers();
for (i = 0; i < classifiers.length; i++) {
   classifier = classifiers[i];
   if ("test".equalsIgnoreCase(classifier.getValue())) {
      emsg = new ExternalMessage();
      emsg.setDefaultMessage("Unallowed Value");
      ValidationMessage vmsg =
                  ValidationMessage.createClassifierMessage(
                  ValidationMessage.RESULT_TYPE_SEVERE, emsg,
                  classifier.getName(), classifier.getValue() );
      results.addValidationMessage(vmsg);
   }
}
```

Validation Scripts

SOA Software predefines a number of validation scripts in the "scripts" directory of the Document Repository:

ValidateAssetName.bsh

The ValidateAssetName script is used in default process-configurations to warn users when they are creating an asset with a name and version matching that of another asset. The default configuration for this script is:

```
  <asset-validator name="CheckAssetName"
                   class="com.logiclibrary.registry.ScriptValidator">
      <properties>
        <property name="script-id" value="scripts/ValidateAssetName.bsh" />
      </properties>
```

```
    <validation-context>ASSET_CREATION</validation-context>
    <validation-context>ASSET_UPDATE</validation-context>
  </asset-validator>
```

ValidateAssetNameType.bsh

This is a variation of the ValidateAssetName validation script that issues a warning when an asset's name, version *and* asset-type match that of another asset. This script is not used in the default configuration.

ServiceNamespaceValidator

- **Behavior:**
  Before a service is published, the validator can warn users of potential namespace conflicts with already published assets. This validator will check an asset's WSDL against other WSDLs to see if there are already existing assets containing a service, port type, or binding with the same qualified name (namespace + local name). It will display any conflicts back to the user in one of 3 configurable severities.

  For Lifecycle Manager to search existing WSDLs, the corresponding WSDL artifact category (default: message-definition) must be made queryable in the Global Definition Template.
- **Class:** com.logiclibrary.validators.ServiceNamespaceValidator
- **Properties:**
  - *severity*
    This property specifies the validation error message severity. It can be either "info", "warning", or "error". The default is "error".
  - *service-artifact-category*
    This is the artifact category containing the service (WSDL). The default is "message-definition"
  - *packed-service-artifact-category*
    This is the artifact category containing a packed service (ZIP). The service and relevant schemas can be packaged together in this artifact. The default is "packed-service"
  - *service-namespace-classifier*
    The namespace classifier that contains the target namespace of the service. This is used to resolve services that are located within Lifecycle Manager. The default is "target-namespace".
  - *service-name-classifier*
    The name of the classifier that contains the service name that the asset represents. Multiple services within a WSDL will cause errors if this classifier is not set. The default is "service-name".
  - *schema-artifact-category*
    This is the artifact category containing schemas. The default is "schema-definition"

- *schema-namespace-classifier*
  The namespace classifier that contains the target namespace of the schema. This is used to resolve schemas that are located within Lifecycle Manager. The default is "target-namespace".

SubsidiaryAssetValidator

- **Class:** `com.logiclibrary.registry.SubsidiaryAssetValidator`
- **Behavior:**
  Coordinates the lifecycle of subsidiary assets with their parent asset. This includes locking, deletion, submission, copying and naming of subsidiary assets. The specific behaviors are as follows:
  - **Locking**
    Subsidiary assets are locked and unlocked when their parent is unlocked.
  - **Deletion**
    Subsidiary assets are deleted when their parent is deleted or when the relationship to the subsidiary asset is removed from the parent asset.
  - **Submission**
    Subsidiary assets are submitted when their parent is submitted.
  - **Copying**
    Subsidiary assets are copied when their parent asset is copied using "create like" or "create new version".
  - **Naming**
    The version of a subsidiary asset will be set to its parent asset's version on the initial save of the subsidiary asset. Optionally, the validator can automatically prefix subsidiary asset names with the name of the parent asset.
- **Usage Context:**
  This validator should be declared whenever subsidiary asset types are used. It should be associated with the ASSET_CREATION, ASSET_NEW_VERSION_CREATION, ASSET_LIKE_CREATION, and ASSET_UPDATE contexts.
- **Properties:**
  - *manage-subsidiary-names*
    A Boolean property that indicates that subsidiary asset names should be prefixed with the name of their parent asset using the format "<parent name>:<subsidiary name>". This is an optional property and defaults to "true".

WSDLValidator

- **Class:** `com.logiclibrary.registry.WSDLValidator`
- **Behavior:**
  Validates various aspects of a WSDL artifact including whether or not include/import references can be resolved and whether the namespace or service name classifiers are invalid (if they exist).
- **General WSDL Properties:**
  - *service-artifact-category*
    This is the artifact category containing the service (WSDL). The default is "message-definition"

- *packed-service-artifact-category*
  This is the artifact category containing a packed service (ZIP). The service and relevant schemas can be packaged together in this artifact. The default is "packed-service"
- *service-namespace-classifier*
  The namespace classifier that contains the target namespace of the service. This is used to resolve services that are located within Lifecycle Manager. The default is "target-namespace".
- *service-name-classifier*
  The name of the classifier that contains the service name that the asset represents. Multiple services within a WSDL will cause errors if this classifier is not set. The default is "service-name".
- *schema-artifact-category*
  This is the artifact category containing schemas. The default is "schema-definition"
- *schema-namespace-classifier*
  The namespace classifier that contains the target namespace of the schema. This is used to resolve schemas that are located within Lifecycle Manager. The default is "target-namespace".
- *missing-governed-assets-severity*
  This property controls whether to flag any resources (services/schemas) that are resolved externally. The purpose is to enforce that any used resources are resolved by namespace from within Lifecycle Manager. This property can take the values of "off", "info", "warning", or "severe". The default is "off".
- *unresolved-resources-severity*
  This property controls whether to display warnings about missing imported/included schemas it can't find. This property can take the values of "off", "info", "warning", or "severe". The default is "severe".
- *unmatched-relationship-severity*
  This property controls whether to display messages for WSDL and schema relationships that exist on the asset but don't correspond with actual dependencies in the WSDL or schema artifacts. This property can take the values of "off", "info", "warning", or "severe". The default is "warning".
- *populate-classifiers*
  This Boolean property controls whether to add the namespace, location (if the WSDL is by reference), version, and binding classifiers to the asset. Valid values are "true" or "false". Default: true.
- *overwrite-classifiers*
  This Boolean property controls whether to overwrite the namespace, location classifier (if the WSDL is by reference), version, and binding classifiers on the asset. Valid values are "true" or "false". Default: false.
- *create-relationships*
  This Boolean property controls whether to add relationships to resources referenced by the wsdl. Valid values are "true" or "false". Default:false.

- *referenced-wsdl-relationship*
  This property defines the relationship type for WSDLs imported by the asset's wsdl. Default: imports.
- *referenced-schema-relationship*
  This property defines the relationship type for schemas imported by the asset's wsdl. Default: imports.
- *included-schema-relationship*
  This property defines the relationship type for schemas included by the asset's wsdl. Default: includes.
- *unpublished-related-asset severity*
  If a related asset isn't published when a relationship is created, this property will define the severity of the unpublished asset message that is displayed to the user. *Valid values are "*off*",* "info", "warning", or "severe". *Default:* severe.

XMLValidator

- **Class:** `com.logiclibrary.registry.XMLValidator`
- **Behavior:**
  Performs XML validation (including schema validation) on an XML artifact. This validator flags malformed XML files and files not conforming to their schemas.
- **Properties:**
  - *severity*
    Any problems validating the artifact will be flagged with this severity. Valid options include "info", "warning", or "severe". Default: warning
  - *target-artifact-category*
    The artifact category containing the XML files to validate. Required.
  - *<XML namespace>*
    An arbitrary number of namespace-to-schema mappings may be added using the XML namespace name as the property key and a URL to the associated schema as the associated property value. For example:
    ```
    <property name="http://www.w3.org/2001/XMLSchema"
    value="http://www.w3.org/2001/XMLSchema.xsd" />
    ```
    The default namespace (noNamespace) schema is specified using the key "noNamespaceSchema".
    Any namespace referenced in the XML artifacts to be validated, which does not correspond to a schema mapping, will result in a validation failure. Note that schema mappings for basic XML, SOAP, and WSDL documents are defined by default and do not need to be specified as properties. These default mappings are as follows:
    - Name: http://schemas.xmlsoap.org/wsdl/http/
      Value: http://schemas.xmlsoap.org/wsdl/http/
    - Name: http://schemas.xmlsoap.org/wsdl/soap/
      Value: http://schemas.xmlsoap.org/wsdl/soap/
    - Name: http://www.w3.org/2001/XMLSchema
      Value: http://www.w3.org/2001/XMLSchema.xsd
    - Name: http://schemas.xmlsoap.org/soap/encoding/
      Value: http://schemas.xmlsoap.org/soap/encoding/

- Name: http://schemas.xmlsoap.org/wsdl/mime/
  Value: http://schemas.xmlsoap.org/wsdl/mime/
- Name: http://schemas.xmlsoap.org/wsdl/
  Value: http://schemas.xmlsoap.org/wsdl/

Request Validators

The following Request Validator classes are currently available:

ScriptRequestValidator

- **Class:** `com.logiclibrary.registry.ScriptRequestValidator`
- **Behavior:**
  Implementation of RequestValidator that invokes a beanshell or jython script.
- **Properties:**
  - *script-id*
    Name of the script that is executed to validate the request. This property must be specified. This document must be stored in the Configuration Document Repository of the library for which the class is configured (see Appendix O for details).
  - *script-type*
    Type of the script. This can either be set to "beanshell" or "jython". This parameter is optional, as the type of script will be inferred from the script-id if possible.
- **Variables:**
  There are several variables that are set prior to the script being run. These variables pass in certain information about what is being validated as well as handle validation messages to display to the user.
  - *request* (com.soa.repositorymanager.library.AssetRequest) in/out
    The request object from which to retrieve or update properties. The request object can be modified in the script (e.g. to set the value of a property)
  - *context* (com.soa.repositorymanager.external.Context) in
    This object contains context information about the request being validated: library, user, etc.
  - *validationResults* (com.soa.logiclibrary.external.ValidationMessages) in/out
    This object is a container that holds the validation messages generated during the script's execution. These messages will be displayed to the end user after successful completion of the script.
  - *configurationProperties* (java.util.Properties) in
    These are the properties set in the request-validator element in the LPC.

The validation script can make use of any APIs exposed through the extensions.zip package.

Appendix R: Artifact Transforms

XSLTransform

- **Behavior:**
  Transforms artifact contents based on an XSL transform document stored in the Document Repository.
- **Usage Context:**
  Generally used to transform the contents of an artifact to a more human-readable form (e.g. HTML). May be used with Lifecycle Manager Artifact Source as a target artifact-source when the transform is to be applied to another artifact on the same asset. SOA provides a transform utilizing this class in the default configuration to allow WSDL artifacts to be transformed to HTML for viewing.
- **Properties:**
  - *transform-document-id*
    The id of the XSL document in the Document Source.
  - *content-type*
    ContentType (MIME type) to be specified to the UI for the output document (e.g. "text/html"). If not specified, the content-type of the source artifact will be used. This field will determine how the UI displays the output of the transform.
  - *file-extension*
    The file extension (e.g. ".html") to be used for the output document. If not specified, the extension of the source artifact will be used.

Appendix S: Asset Search Customization

The Lifecycle Manager and Portfolio Manager products support customizable queries when deployed on Oracle[50] or SQLServer databases. This feature allows administrators to design custom queries for use by Lifecycle Manager and Portfolio Manager end users. The custom query support consists of two primary concepts "Search Criteria" and "Searches" as described in the following sections:

Search Criteria

Search Criteria represent the individual conditions that must be met for an asset to match a search. Search Criteria are generally defined using XPath expressions over XML representations of assets and artifacts. There are three possible "targets" for search criteria:

- **ASSET**
  The ASSET target indicates that the XPath expression is to be applied to the XML representation of the assets in the library[51].
- **XML_ARTIFACT**
  The XML_ARTIFACT target indicates that the XPath expression is to be applied to specified artifact contents in XML format. Such artifacts include WSDLs or BPELs.

[50] Oracle Text search support must be enabled

[51] The schema for the asset XML documents exists in the Document Repository and can be retrieved using the GetDocument command passing "asset.xsd" as the document Id. It is also possible for usage controllers to view the XML representation of an asset in the library using the [Show XML] link on the asset detail page.

- **ARTIFACT**
  This target is used to indicate that the contents of designated queryable artifacts are included in the search. This type of criteria always relies on the full-text search capability of the underlying database and does not use an XPath expression.

Search Criteria targeted at artifacts may additionally specify the target artifact category and name, thus further restricting the search results.

When defined in the process configuration document, a unique name must be provided for each search criteria to allow referencing from a search declaration. Search criteria may optionally specify a display name and description which are used when a user views search criteria from the search details page.

Xpath Expressions

In the case of search-criteria targeted at ASSET and XML_ARTIFACT, a valid XPath expression[52] must be provided in the search-criteria declaration.

Evaluation Style

Two styles of XPath evaluation are supported by the product and are designated in the "evaluation-type" attribute in the search-criteria element. The supported evaluation types are as follows:

HAS_PATH

This evaluation approach requires that queried XML documents contain the path specified in the XPath expression. Thus any comparison logic needs to be contained within the XPath expression. For example, the XPath expression `/asset[version="1"]` will match only documents where the asset's version is set to "1". This approach is recommended when comparing to a known literal (e.g. "1") since it uses the comparison semantics of the XPath specification.

IN_PATH

This evaluation approach requires that the XPath expression specify a path to a text node within the document. The search phrase from the containing search (entered by the user) is then checked for containment within the specified node(s). For example, the XPath expression

`/asset/description`

may be used to check the asset's description for containment of the search phrase. The containment check in most cases (see table below) will utilize the full-text search semantics of the underlying database. This means that matching will be at a word level and will take into account stop-words, synonyms, etc. Logical terms such as AND/OR/NOT are also supported. See your database documentation for additional details on full-text search semantics.

Fields and Substitution

[52] Expressions are checked for syntactical correctness on upload of the process configuration document.

In certain cases it may be necessary to access user input from within a HAS_PATH style XPath expression. This is supported through the use of parameter substitution within the XPath string. For example, the following Xpath expression

```
/asset[name = "CurrencyExchangeService"]
```

may be written using parameter substitution as follows to indicate that the user-input search phrase from the containing search be used as the literal:

```
/asset[name ="{search.field.GLOBAL}"]
```

Now, if the user inputs the phrase "Customer" when running the search, the actual expression evaluated will be:

```
/asset[name = "Customer"]
```

Substitution parameters must occur as literals in the XPath expression and be wrapped with '{' and '}' braces. The text within the braces must specify one of:

- "search.field.GLOBAL"
  Indicates the primary user-entered search phrase from the containing search
- "search.field.<field name>"
  Where <field name> is the name of a search-field provided in a <field> child element of the search-criteria declaration[53].

When used in an equality comparison, substitution will honor the AND/OR/NOT semantics of the search phrase. Using the example expression above, if the user had entered "Customer or CurrencyExchangeService" the resolved expression would be:

```
/asset[name = "Customer" or name = "CurrencyExchangeService"]
```

Note that evaluation of equality follows the XPath semantics and will be case-sensitive.

Defaultable Fields

A search-field may be marked as "defaultable" to indicate that the search-criteria can tolerate a missing value for the field. In this case, the specific predicate in the XPATH criteria will be ignored. If a field not marked as defaultable has an empty value, the entire XPATH criteria will be ignored. For example, consider the following XPATH expression is:

```
/asset[name ="{search.field.asset_name}" and version
="{search.field.asset_version}"]
```

and assume the user does not enter a value for the field "asset_version". If defaultable is false (the default value), the entire criteria will be skipped. If defaultable is true, the criteria will become:

```
/asset[name ="{search.field.asset_name}"]
```

Namespaces

The schema used for the asset XML does not designate a default namespace. This allows XPath expressions to be written without regard for namespaces or prefixes. However, it may be the case with XML artifacts that elements are namespace qualified. In such cases, the XPath expression must be written with namespace prefixes, for example:

```
/ns1:department/ns1:employee[@name="Fred"]
```

The prefix must then be mapped to the appropriate namespace URI by adding a <namespace> element to the <xpath-criteria> element. For example:

```
<namespace prefix="ns1" uri="http://some_company.org/schema/" />
```

[53] Fields other than the GLOBAL_SEARCH_FIELD are not currently supported by the RM/PortM user interfaces.

Note that as a convenience, the Lifecycle Manager and Portfolio Manager products strip standard namespaces when indexing WSDL artifacts, eliminating the need qualify standard elements such as "operation" in the XPath expressions. The following namespaces are removed for indexing:
http://schemas.xmlsoap.org/wsdl/
http://www.w3.org/2001/XMLSchema/
http://schemas.xmlsoap.org/wsdl/soap/
http://schemas.xmlsoap.org/wsdl/http/
http://schemas.xmlsoap.org/wsdl/mime/
http://schemas.xmlsoap.org/wsdl/soap12/
http://schemas.xmlsoap.org/soap/encoding/

Database and Performance Considerations

The XPath search feature uses the XML and full-text indexing support of the underlying database. The evaluation-style chosen and the content of the XPath expression will impact the performance of containing searches; which also varies by database.
The database's full text indexing will normally be used when the XPath expression meets these criteria:

- The expression does not contain non-string comparisons (such as ">" or "<").
- The expression does not use XPath functions (such as "contains()").
- The expression does not compare XML attributes (SQLServer-only restriction)[54].

The following tables provide evaluation details by evaluation-type and text compatibility:

	Oracle/DB2[55]		SQLServer	
	HAS_PATH	IN_PATH	HAS_PATH	IN_PATH
Text compatible	Uses text indexing with proportional scoring[56].	Uses text indexing with proportional scoring.	Uses text indexing with proportional scoring.	Uses text indexing with Boolean scoring.
Not Text compatible	Text indexing not used, Boolean scoring	Not supported[57]	Text indexing not used, Boolean scoring.	Text indexing not used, Boolean scoring[58].

[54] SQLServer does not text-index XML attributes, however Oracle does.
[55] While DB2's behavior is similar to Oracle's, DB2 adds the restriction that wildcards ('*') are not supported in XPath expressions.
[56] "proportional" scoring indicates a score determined by text index based on best match, while "Boolean" scoring indicates the score will be either 0 or 100 depending on compliance with the expression.
[57] Consider using HAS_PATH with substitution if expression requires user input.
[58] In this case the SQL "LIKE" operator is used to check for containment of the search phrase in the specified path, meaning that partial-match semantics are supported.

Searches

Asset searches defined in the process configuration document serve as search "types" for Lifecycle Manager and Portfolio Manager users. Searches require a unique name and allow for an optional description. Users search for assets by selecting one of the predefined searches by name, optionally customize the search with classifier filtering, and finally run the search providing a search phrase.

Search Criteria Associations

The behavior of a search is dependent upon the search criteria associated with that search. These associations are defined in <search-criteria-association> elements. In addition to the referenced search-criteria name, compliance semantics and a weighting factor may optionally be specified.

Compliance

The "compliance" attribute allows one of three choices:

- MUST_COMPLY
  This option indicates that compliance with the referenced search-criteria is required. Any asset not complying with the referenced search-criteria are omitted from the search results.
- MUST_NOT_COMPLY
  This option is the logical inverse of MUST_COMPLY: only assets not complying with the referenced search-criteria are included in the search results.
- MAY_COMPLY
  This option is used to specify that the referenced search-criteria is optional and affects only the score of the assets in the search results. The referenced criteria will not cause assets to be omitted from the search results[59]. This is the default compliance setting when the "compliance" attribute is not specified.

Required (MUST_COMPLY, MUST_NOT_COMPLY) criteria may be combined with optional (MAY_COMPLY) criteria within a search. The default behavior in this case is to consider assets as matching when they comply with the required criteria regardless of whether they comply with any of the optional criteria. The optional criteria in this case serve only to increase the score of the matching assets. The Boolean attribute "optional-criteria-match-required" on the asset-search elements may be used to require that assets match at least one optional criteria in addition to the required criteria. This is useful when a required criteria is used to serve as a filter for a search that would otherwise contain only optional criteria.

Weight and Scoring

The "weight" attribute allows for tailoring of the impact on the overall asset score that a specified search-criteria will have. The resulting score from an XPath criteria is based on the database indexing approach used and the characteristics of the XPath criteria (see table above). Because of this variance, the Lifecycle Manager and Portfolio Manager products use scoring

[59] In cases where a search consists only of MAY_COMPLY search-criteria, result assets must comply with at least one of the referenced search-criteria. This prevents such a search from automatically returning all assets in the library.

only as a means to provide a "best match" ordering of results without exposing the actual score (which does not have a fixed upper bound) to the end user.

Asset Filter

The <asset-filter-name> attribute of the search definition may optionally be used to specify an <asset-filter> defined in the global <asset-filters> element of the process configuration document. The specified asset-filter will be used to filter the results of the search according to the classifier criteria in the filter. See the Asset Filters section of this guide for additional details.

Property Filter

The <property-filter-name> attribute of the search definition may optionally be used to specify a <property-filter> defined in the global <property-filters> element of the process configuration document. The specified property-filter will be used to filter the results of the search according to the property criteria in the filter. See the Property Filters section of this guide for additional details.

Search Visibility

Searches defined in the process configuration document are made visible by default, meaning they will appear as a search type choice to end users. It is possible to hide a search from end-users through the use of the "visible" attribute. Setting this attribute to "false" will hide the search.

Search Type

A published-asset-search is used for querying published assets while a catalog-asset-search is used to query assets in progress. Catalog searches may additionally specify these attributes:

- *published* – Boolean used to specify that only published or not published assets should be returned
- *in-progress* – Boolean used to indicate that only assets with changes in progress or no changes in progress should be returned.
- *accessible-to* – Used to specify a user name. Only assets accessible to the specified user (as an ACE) in the catalog will be returned.
- *changed-by* – Used to specify a user name. Only assets changed by the specified user will be returned.

Predefined Searches

The Lifecycle Manager and Portforlio Manager products come with a number of "built-in" searches that will be present even if not explicitly defined in the process configuration document. These predefined searches use the names:

- "Assets by Content"
- "Assets by Name"
- "Assets by Keyword"

The behaviors of these searches may be overridden by explicitly defining them in the process configuration document. Such a definition will override the predefined definition. It is also possible to hide a predefined search from end users. This is accomplished by explicitly defining the search in the process configuration document and specifying the "visible" attribute on the search as "false".

Default Search

The "Assets by Content" search is automatically designated as the "default" search, indicating that it is the search used when the "search" button in the left navigation pane is clicked. It is possible to explicitly define another search as the default by naming it "Default Assets". The look-up order for the default search is as follows:

1. Check for a search named "Default Assets"
2. If 1 not found, check for a search named "Assets By Content"
3. If 2 not found use the default definition for "Assets By Content"

Any search designated for use as the default search must require only the global search-field.

Related Asset Searches

The searches used to show an asset's related assets as well as candidate assets for a relationship are also customizable. The purpose of customizing related asset searches is normally to add context based filtering criteria, commonly to limit the returned assets to those that comply with some visibility rule.

For published assets, related asset searching uses an implicit predefined search called "Default Published Related Assets". This search may be explicitly defined in the LPC to add filtering criteria.

Candidate related asset searches may be defined for both in-progress assets as well as published assets[60]. "Default Published Related Asset Candidates" and "Default Catalog Related Asset Candidates" are the implicit default searches respectively for published and in-progress assets. When a relationship is defined in the GDT using the <define-asset-relationship> element, it is also possible to specify custom searches to be used to find candidates specifically for that relationship. This is done using the attributes "published-asset-search" and "catalog-asset-search". The searching of in-progress candidate assets for a relationship may be disabled by setting the "catalog-asset-search" attribute to "DISABLED". The look-up algorithm for these searches is as follows:

For published asset candidates:

1. If "published-asset-search" specified in <define-asset-relationship>, use this search

[60] Note that catalog asset searches used for related asset search will have "published" implicitly set to "false" when executed. This is to provide a complimentary view of the assets returned by the published asset search.

2. Otherwise look for an explicitly defined search called “Default Published Related Asset Candidates”
3. Otherwise use the implicitly defined search for published related assets

For in-progress asset candidates:

1. If “catalog-asset-search” specified as “DISABLED” in <define-asset-relationship> do not allow in-progress candidate search.
2. Otherwise, if a search is specified in the “catalog-asset-search” attribute, use this search
3. Otherwise look for an explicitly defined search called “Default Catalog Related Asset Candidates”
4. Otherwise use the implicitly defined search for catalog related assets

Note that when customizing candidate searches, the fields used by published and in-progress searches should be common, allowing the UI to switch between published and in-progress results without requiring additional user input. The default candidate asset searches require a single asset name search field.

Search Style Settings

The Lifecycle Manager and Portfolio Manager products continue to support the “traditional” non-customizable search approach on all platforms, while support for customizable search varies by database as follows:

- **SQLServer**
  Customizable Search enabled by default.
- **Oracle**
  Oracle installations may be configured to run with or without text indexing. Text support can be enabled through the use of the EnableTextSearchOnOracle command (see Administration Guide for additional details).
  If text support is disabled, only traditional search support is available.
  If text support is enabled, Customizable Search is enabled by default.
- **UDB**
  Customizable Search enabled by default as of release 6.3.1.

Search style may be configured on SQLServer and Oracle installations by using the SetInstallationProperty command to set the property “SOA_QUERY_TYPE” to either “QUERY_TYPE_STANDARD” for traditional search style or “QUERY_TYPE_ADVANCED” for Customizable Search style.

TestXPathCriteria Command

The TestXPathCriteria command may be used to validate <xpath-criteria> elements before the configuration document is deployed to the library. The command validates the XPath syntax, checks for text index compatibility, and finally runs the criteria in a test search against the specified library. The parameters to the command are:

Parm 1: < xpath-criteria> in XML format (as it would appear in the process configuration document)

Parm 2: A search phrase to be used in the test search when the <xpath-criteria> element specifies IN_PATH compliance.

Customizing Asset Tree Search

In libraries with advanced search enabled (QUERY_TYPE_ADVANCED) it is possible to customize the search criteria used in the search that produces the asset tree. This is accomplished by explicitly defining a published-asset-search in the LPC with the name of "Default Tree". This search is then used as the base search in producing the asset tree. Explicit definition of the default tree search is most commonly used to add filtering criteria to the asset tree[61]. Here is an example that overrides the default tree search to show only "Production" assets:

In the search-criteria section:

```
<xpath-criteria name="ProductionAssets"
                evaluation-type="HAS_PATH"
                target="ASSET">
   <expression>/asset/classifiers/classifier[name="status" and values/value
="Production"]</expression>
</xpath-criteria>
```

In the searches section:

```
<published-asset-search name="Default Tree"
                        visible="false"
                        description="Search only for assets with status =
                        Production">
   <search-criteria-association
      search-criteria-name="ProductionAssets"
      compliance="MUST_COMPLY"
      weight="1" />
</published-asset-search>
```

Note that the "visible" attribute is set to "false"; this prevents the search from being seen in the search selection box of the user interface.

Search Configuration Examples

Assets by Keyword

This example shows the default configuration of the predefined "Assets By Keyword" search: First, add an <xpath-criteria> element to the search-criteria section of the configuration document:

```
<xpath-criteria
   name= "KeywordMatch"
   evaluation-type="IN_PATH"
   target="ASSET" display-name="Classifier keyword Value Match"
   description="The value of the keyword classifier matches tokens in the
search string">
   <expression>/asset/classifiers/classifier[name="keyword"]/values</expressi
on>
</xpath-criteria>
```

[61] Note that if not explicitly defined, the default tree search will contain no search-criteria and will return all assets.

This declaration specifies an xpath-criteria named “KeywordMatch” that will be evaluated using the “IN_PATH” style, meaning that the search phrase of the containing search will be checked for containment within classifier value elements with the name “keyword”.

Next, define the search:

```
<published-asset-search
   name="Assets by Keyword"
   visible="true"
   description="Search for Assets by keyword classifier">
   <search-criteria-association
      search-criteria-name="KeywordMatch"
      compliance="MUST_COMPLY"
      weight="1" />
</published-asset-search>
```

Here a published-asset-search is defined with a single search-criteria-association referencing the “KeywordMatch” criteria shown above. The compliance is set to MUST_COMPLY, meaning that only assets that match the criteria (using full-text semantics) will be included in the results. Setting the weight to “1” simply allows the standard scoring from the search-criteria to be used. The choice of weighting factor here is not particularly interesting since there is only one search-criteria-association.

Assets by Name

This example shows the default configuration of the predefined “Assets By Name” search. Here is the search-criteria:

```
<xpath-criteria
   name="AssetNameMatchCriteria"
   evaluation-type="IN_PATH"
   target="ASSET"
   display-name="Asset Name Match"
   description="Asset name matches tokens in the search string">
   <expression>/asset/alias</expression>
</xpath-criteria>
```

Note that The AssetNameMatchCriteria is actually using the “/asset/alias” path. This is because the alias element contains the asset name as well as a tokenized representation of the name resulting from a separation on lower-to-upper case changes as well as common delimiters such as “_”. Using the alias element allows, for example, the asset with name “CurrencyConverter” to match on the search string “currency”.

The search definition:

```
<published-asset-search
      name="Assets by Name"
   visible="true"
   description="Search for Assets by name allowing partial matches">
      <search-criteria-association
            search-criteria-name=" AssetNameMatchCriteria "
      compliance="MUST_COMPLY"
      weight="1" />
</published-asset-search>
```

Assets by Content

This example shows the default configuration of the all-purpose "Assets by Content" search. Here are the search-criteria:

```
<xpath-criteria
   name="MetadataMatchCriteria"
   evaluation-type="IN_PATH"
   target="ASSET"
   display-name="General Metadata Match"
   description="Any asset content matches tokens in the search string">
   <expression>/asset</expression>
</xpath-criteria>
<xpath-criteria
   name="AnyArtifactMatchCriteria"
   evaluation-type="IN_PATH"
   target="ARTIFACT"
   display-name="Artifact Content Match"
   description="The asset contains a queryable artifact whose contents
matches tokens in the search string">
</xpath-criteria>
```

Notice the following:

- The MetadataMatchCriteria covers all basic asset metadata content.
- The "AnyArtifactMatchCriteria" applies to the "ARTIFACT" target and does not require an <expression> element. The addition of this criteria to the search will result in the contents of queryable artifacts being searched using full-text indexing.

The Search definition:

```
<published-asset-search
   name="Assets by Content"
   visible="true"
   description="Token-match search for Assets including metadata and Artifact
content">
   <search-criteria-association
      search-criteria-name="MetadataMatchCriteria"
      compliance="MAY_COMPLY"
      weight="10" />
   <search-criteria-association
      search-criteria-name="AnyArtifactMatchCriteria"
      compliance="MAY_COMPLY"
      weight="1" />
</published-asset-search>
```

Notice in the search definition that all search-criteria-associations are "MAY_COMPLY" compliance meaning that at least one must match (result in a nonzero score).). The weighting factors are also adjusted to favor asset metadata match and reduce the impact of artifact content match. Note that the Assets By Content search could be further optimized by specifying only the single MetadataMatchCriteria and placing AnyArtifactMatchCriteria into a separate search.

Services by WSDL Operation Name

This example shows the construction of a search that can be used to find Service assets with a particular WSDL operation name[62].
Here is the xpath-criteria:

```
<xpath-criteria
      name="WSDLsByOperation"
      evaluation-type="IN_PATH"
      target="XML_ARTIFACT">
     <expression>/definitions/portType/operation/@name</expression>
</xpath-criteria>
```

The containing search definition:

```
<published-asset-search
      name="Services by Operation"
      visible="true"
   description="Search for Service assets by operation name">
   <search-criteria-association
            search-criteria-name="WSDLsByOperation"
            compliance="MUST_COMPLY"
      weight="1" />
   <asset-filter-name>service-assets</asset-filter-name>
</published-asset-search>
```

Notice that in addition to the search-criteria-association, an Asset Filter has been specified to limit result assets. It's assumed that the "service-assets" filter would restrict the "asset-type" classifier to "Service".

Common XPath Expressions

The following is a list of useful XPath expressions for customizable searches..

Assets by specified classifier value and user-provided value:

```
/asset/classifiers/classifier[name="a-classifier-name"]/value
```

Target: ASSET Style: IN_PATH
Note: By switching to HAS_PATH this expression can be used to find assets having or not having a value for the specified classifier, depending on the choice of compliance

Assets having specified relationship type :

```
/asset/related-assets/related-asset[name="a-relationship-name" and
incoming="false"]
```

Target: ASSET Style: HAS_PATH
Note: Switching "incoming" to "true" would produce assets that are the targets of the specified relationship.

Assets having specified artifact type :

```
/asset/artifacts/artifact[category="an-artifact-category"]
```

Target: ASSET Style: HAS_PATH

[62] Note that the SQLServer database does not support text indexing of wsdl documents in its default configuration. A Full Text Search Filter must be added to the database to handle wsdl files. See Microsoft SQLServer documentation for more information.

Assets within the active Project's path:
This criteria uses context parameter substitution to restrict matching assets to those with owning-group set to a group in the active project's ancestor path.
`/asset/owning-group[id="{project.path}"]`
Target: ASSET Style: HAS_PATH

WSDL documents by operation name

`/definitions/portType/operation/@name`Target: XML_ARTIFACT Style: IN_PATH

Simple Asset Element Paths
The list that follows are simple paths to elements within the asset schema:
/asset/name
/asset/version
/asset/description
/asset/locking-user (catalog asset searches only)
/asset/last-user
/asset/internal-version
/asset/created-date
/asset/modified-date
/asset/owning-group/name
/asset/owning-group/id
/asset/owning-group/path
/asset/context-project/name
/asset/context-project/id
/asset/context-project/path
/asset/properties/property
/asset/properties/property/name
/asset/properties/property/display-name
/asset/properties/property/type
/asset/properties/property/values
/asset/classifiers/classifier
/asset/classifiers/classifier/name
/asset/classifiers/classifier/display-name
/asset/classifiers/classifier/type
/asset/classifiers/classifier/values
/asset/artifacts/artifact
/asset/artifacts/artifact/name
/asset/artifacts/artifact/display-name
/asset/artifacts/artifact/category
/asset/artifacts/artifact/containment
/asset/artifacts/artifact/reference
/asset/artifacts/artifact/id
/asset/artifacts/artifact/version-info
/asset/artifacts/artifact/file-name

/asset/artifacts/artifact/file-extension
/asset/related-assets/related-asset
/asset/related-assets/related-asset/name
/asset/related-assets/related-asset/display-name
/asset/related-assets/related-asset/related-asset-id
/asset/related-assets/related-asset/related-asset-name
/asset/related-assets/related-asset/related-asset-version
/asset/related-assets/related-asset/incoming
/asset/related-assets/related-asset/properties (relationship-context properties)
/asset/related-assets/related-asset/properties/property
/asset/related-assets/related-asset/properties/property/name
/asset/related-assets/related-asset/properties/property/display-name
/asset/related-assets/related-asset/properties/property/type
/asset/related-assets/related-asset/properties/property/values

Custom Search Examples

Hiding Withdrawn Assets

It is common that a library supports an asset lifecycle state for withdrawn or retired assets. It's possible to modify the basic searches to skip assets in this state.

The first step is to define a search-criteria that specifies the state to be omitted. Assuming the actual status classifier value to omit is "Withdrawn" the search-criteria definition would look like:

```
<xpath-criteria
      name="Not Withdrawn"
      evaluation-type="HAS_PATH"
      target="ASSET"
      display-name="Not Withdrawn"
      description="Show only assets that are not withdrawn from use">
      <expression>/asset/classifiers/classifier[name =
"status"]/values[value!="Withdrawn"]</expression>
</xpath-criteria>
```

Note that the evaluation-type of the expression is "HAS_PATH", indicating that this criteria is checking for existence of the XPATH in the asset. In this case, assets with a status classifier value other than "Withdrawn" will match.

The next step is to add the new criteria to the existing searches. Here is the default "Assets by Context" search modified to include the new criteria:

```
<published-asset-search
      name="Assets by Content"
      visible="true"
      description="Search for Assets by metadata and artifact content"
      optional-criteria-match-required = "true">
      <search-criteria-association
            search-criteria-name="MetadataMatchCriteria"
            compliance="MAY_COMPLY"
            weight="10" />
      <search-criteria-association
```

```
            search-criteria-name="AnyArtifactMatchCriteria"
            compliance="MAY_COMPLY"
            weight="1" />
      <search-criteria-association
            search-criteria-name="Not Withdrawn"
            compliance="MUST_COMPLY"
            weight="1" />
</published-asset-search>
```

Note that the weight attribute on the "Not Withdrawn" criteria does not matter as any asset returned from this search will get full credit for this mandatory criteria.
Also notice the use of the optional-criteria-match-required attribute. If this attribute were "false" (the default value) any asset complying to the "Not Withdrawn" criteria would come back in the search results regardless of whether it matched either of the two optional criteria. Setting the attribute to true ensures that only assets that match the mandatory criteria and at least one of the optional criteria are returned.

Finding Assets Modified Since a Specified Date

Assets in the library store a "modified-date" attribute in their queryable XML record indicating the last date (and time) that the asset was published. This allows a search to be written to return assets modified since a specified date. Here is the search-criteria definition:

```
<xpath-criteria
      name="ModifiedSince"
      evaluation-type="HAS_PATH"
      target="ASSET"
      display-name="Modified since"
      description="Assets modified since the specified date">
      <field
            name = "modified-date"
            type = "DATE"
            display-name = "Modified Date" />
     <expression>/asset[translate(modified-date,"-:TGMT","0")>=
translate("{search.field.modified-time}000000000000","-","0")]</expression>
</xpath-criteria>
```

This criteria makes use of a date entry field called "modified-date" declared in the nested <field> element. The type of "DATE" indicates that the UI should use a calendar widget for entry. Within the <expression> the user-entered value for the field is accessed using the replacement parameter "{search.field.modified-date}". Since XPATH-based query does not support advanced functions for date/time manipulation its necessary to adjust the date/time field in the XML record and the user-entered date to allow a numerical comparison.

Here is a dedicated "Modified Since" search definition using the above criteria:

```
<published-asset-search
      name="Modified Since"
      visible="true"
      description="Search for Assets modified since the given date">
      <search-criteria-association
            search-criteria-name="ModifiedSince"
            compliance="MUST_COMPLY" weight="1" />
</published-asset-search>
```

It may be desirable to allow the user to narrow the results by asset name. This is accomplished by combining the ModifiedSince criteria with the standard AssetNameMatchCriteria:

```
<published-asset-search
      name="Modified Since"
      visible="true"
      description="Search for Assets modified since the given date">
      <search-criteria-association
            search-criteria-name="ModifiedSince"
            compliance="MUST_COMPLY" weight="1" />
      <search-criteria-association
            search-criteria-name="AssetNameMatchCriteria"
            compliance="MUST_COMPLY" weight="1" />
</published-asset-search>
```

Appendix T: Exporters

The following Exporter classes are currently available:

ScriptExporter

- **Behavior:**
  A script-based Exporter that runs a BeanShell or Jython script stored in the document repository.
- **Properties:**
  - *script-id*
    Name of the exporter script that is executed. This property must be specified. This document must be stored in the Configuration Document Repository of the library for which the class is configured (see Appendix O for details).
  - *script-type*
    Type of the script. This can either be set to "beanshell" or "jython". This parameter is optional, as the type of script will be inferred from the script-id if possible.
  - Arbitrary properties may also be provided to be passed through to the Exporter script when it's run. These will appear in the "configurationProperties" variable.
- **Variables:**
  The following variables are provided to the script when it is executed.
  - *context*
    Type: com.soa.repositorymanager.external.Context
    Semanitics: IN
    The context variable holds context information in properties. For example, user Id and active project Id.
  - *assetIds*
    Type: java.util.List
    Semantics: IN
    This is a list of selected asset Ids that are to be processed
  - *configurationProperties*
    Type: java.util.properties
    Semantics: IN
    Holds configuration properties provided in the Exporter properties element in the LPC

- *results*
  Type: com.logiclibrary.external.AssetProcessingResults
  Semantics: IN-OUT
  Use this passed in object to add an overall result message as well as per-asset AssetProcessingResult messages. These messages will be displayed by the UI when the exporter is run.

DelimitedFileAssetExporter

- **Behavior:**
  Exports selected assets to a zip file compatible with the DelimitedFileAssetImporter. Please refer to the DelimitedFileAssetImporter for details on the content of the zip file. In addition to the files required by DelimitedFileAssetImporter, the zip file will contain a Readme.htm offering a summary of the export and instructions.

  When importing the assets.txt into a spreadsheet, it may be necessary to select the proper column format for the data. As an example, in Excel you will need to alter the column data format from "General" to "Text". This will avoid the case where Excel would interpret the string "1.0" as a number and automatically change it to "1". A similar behavior can happen with boolean classifier values where "true" will become "TRUE". You can use Excel's importer by starting Excel and then choosing File -> Open. Be aware that double clicking on the "assets.txt" file in File Explorer may immediately open the file into Excel without providing an option to specify the column formats.
- **Properties:**
  - *attributeDelimiter*
    The delimiter character (default is tab). For example, the delimiter between classifiers, artifacts, etc.
  - *attributeValueDelimiter*
    The attribute delimiter separating the values of the attributes (default '^'). Using an artifact as an example, this delimiter would separate the reference, by-reference, version, and optional name.
  - *description*
    The exporter description. (optional)
  - *full-header*
    If false (default), only the columns necessary to support the assets being exported will be created in the header. If true, column headers will be created for every classifier, artifact, and relationship defined in the GDT. (optional)
  - *published-assets-only*
    If true, only published asset content will be exported. If false (default), pending asset content will be included in the export (optional).
  - *export-byval-artifacts*
    If true (default) all by-value artifacts of an asset will be exported. When set to false, by-value artifacts will not be exported. This may be of benefit if assets contain numerous and or large artifacts which are not of interest when exporting.

The 'overview' artifact will always be exported and is unaffected by this property. (optional)

- *export-directory*
  The system property which specifies the temporary directory to use during building of the export file (optional, default=java.io.tmpdir).

Appendix U: UDDI Integration

The Lifecycle Manager UDDI Governance Module allows organizations to automatically control the population of one or more UDDI Registries directly from Lifecycle Manager without human intervention as part of their normal asset publication process. Key features of this module include:

- The ability to establish connections to **multiple UDDI registries** from a **single Lifecycle Manager Library**: this feature enables organizations to establish fanout of published services to multiple registries to support geographically distributed or failover environments. Organizations can also establish multiple registries to support Web service registration within development, testing, and operational environments.
- The ability to create **publication filters** for each established registry connection: organizations can establish publication filters (via the Lifecycle Manager Classification Criteria Set feature) for each configured registry such that specific registries are populated at specific points in the Web service development and deployment lifecycle.
- Support for both **single-WSDL** and **dual-WSDL** (OASIS recommended) Web service definitions: while many Web service tools produce a composite WSDL that combines both service interface declaration and service endpoint deployment information, OASIS recommends a dual-WSDL approach, with one WSDL specifying the service interface and a second WSDL containing deployment details and referring to the interface specification WSDL[63]. The Lifecycle Manager UDDI Governance Module is designed to automatically support both Web service definition modes to provide maximum flexibility to organizations as elaborated upon in the following subsection.
- Support for the publishing of XML schemas into UDDI as "specification" tModels.

Supported Service Asset Types

The Lifecycle Manager UDDI Governance Module supports three types of Web service assets as well as XML schema assets for publication into configured UDDI Registries:

Web service Interface Assets

A web service interface asset represents only the interface of a Web service and contains only a WSDL service interface definition artifact rather than a complete WSDL as its "wsdl" artifact. These assets commonly use the "Service" asset type. When a Lifecycle Manager UDDI Governance Module publishes a service interface asset, a WSDLspec type tModel for the service interface will be created in the target UDDI registry.

[63] For more information on the OASIS approach see http://uddi.org/pubs/wsdlbestpractices-V1.07-Open-20020521.pdf

Web service Implementation Assets

A Web service implementation asset represents a particular implementation of a service defined by a service interface asset. It commonly has an asset type of "Service Implementation" and contains a WSDL document in its "wsdl" artifact that includes service and port definitions. This document must reference a separate service interface definition document. A Web service deployment asset must have an "Implements Service Interface" relationship[64] to its associated web service interface asset.

When a Web service implementation asset is published by the UDDI Governance Module, a business service and binding template are created in the UDDI repository. The binding template is set to reference the tModel associated with the related web service interface asset.

Complete Web service Assets

A complete Web service asset is commonly of type "Service" and will contain a complete WSDL document as an artifact. It is logically the combination of a web service interface and web service deployment asset. Note that this combined approach, while supported by OASIS, is not a recommended OASIS best practice. Such assets will have services and ports defined in their "wsdl" artifacts but will not have ""Implements Service Interface" relationships to other assets. When the UDDI Governance Module publishes a complete Web service asset, it will create the tModel, the business services, and the binding templates defined in the WSDL artifact.

XML Schema Assets

Assets representing non-web service components often will specify an XML interface in an XML Schema artifact. When the Lifecycle Manager UDDI Governance Module publishes such an asset, it may optionally be configured to publish the interface schema into the UDDI registry as a "specification" type tModel. This tModel specifies the schema's target namespace as an identifier and references the actual schema document as the tModel's overview document.

Overview of Configuration Process

At a high level, the Lifecycle Manager UDDI Governance Module configuration process involves the following steps:

- Creating one or more UDDIPublisher entries for the Library
- Optionally establishing filtering criteria to selectively filter Web service Assets for publication to the configured UDDI Registries
- Configuring the Asset types used for "Service" and "Service Implementation" Assets with appropriate metadata definitions to ensure that the necessary organization-specified metadata associated with a Web service is published into the configured UDDI registry

[64] Note that the actual categories of the WSDL artifact and deployment relationship may be customized when configuring a UDDI Publisher through Smart Controls as described in the next section.

Creating a UDDIPublisher Listener

A UDDIPublisher manages the publishing and updating of Web service assets from Lifecycle Manager into a UDDI Registry. It is implemented as a SmartControls Listener that is declared in the Lifecycle Manager Process Configuration document as in the following example[65]:

```
<listener name="TestRegistryPublisher" class="UDDIPublisher">
  <properties>
     <property name="inquiry-url" value="http://westlake/uddi/inquire.asmx" />
     <property name="publish-url" value="http://westlake/uddi/publish.asmx" />
     <property name="default-business-key" value="90e818c1-1ee2-4a69-92df-d698378168cd"/>
     <property name="uddi-user" value="LOGICLIBRARY\tgraser" />
     <property name="uddi-password" value="myPassword" />
     <property name="wsdl-artifact-category" value="service API" />
     <property name="deploys-relationship-category" value="implements" />
  </properties>
</listener>
```

In this example the basic properties of the UDDIPublisher are set to similar values as in the previous UI creation example. However, additional properties "wsdl-artifact-category" and "deploys-relationship-category" may optionally be used to override the default category used for the WSDL artifact ("wsdl") and the default deployment relationship ("deploys") respectively. The property "schema-artifact-category" may be used to enable publishing of XML schema artifacts as tModels in UDDI.

UDDIPublisher Listener Details

- **Behavior:**
  Publishes a web service or schema type Assets to a specified UDDI registry[66].
- **Usage Context:**
  Generally configured to be triggered at Asset publish time by the *ASSET_AUTO_PUBLISH* and *ASSET_MANUAL_PUBLISH* Events. Should be restricted with a Filter allowing only Assets that represent web services, web service deployments or XML schemas.
  NOTE: UDDIPublishers defined in the LPC should not be edited from the existing "Manage UDDI Publishers" page in the support center. This may result in property data being reset.
- **Properties:**
  - *inquiry-url*
    Inquiry URL for the UDDI registry (e.g. "http://westlake/uddi/inquire.asmx")
    This property is mandatory.
  - *publish-url*
    Publishing URL for the UDDI registry (e.g. "http://westlake/uddi/publish.asmx")
    This property is mandatory.
  - *default-business-key*
    UDDI key for the business entity (also known as "provider") that services will be published under (e.g. "87686f01-50f6-4aef-bcae-d363bba848d4"). This business will be used if the web service asset does not specify a business key in the businessKey classifier. The user specified for the UDDIPublisher must have

[65] See the Smart Controls Guide for additional details on configuring Listeners, Filters and Actions.
[66] Additional information about UDDI publication can be found in the *UDDI User's Guide* document

authority in the UDDI registry to add services to the specified business entity. This property is mandatory.
    - *uddi-user*
      The user id to use when publishing to the UDDI registry. All entities created in the UDDI registry will be owned by this user.
      This property is mandatory. If not specified, Lifecycle Manager Application User will be used. (Optional).
    - *uddi-password*
      The password to use when publishing to the UDDI registry.
      This property is mandatory.
    - *wsdl-artifact-category*
      The name of the artifact category that the publisher will use for access to the WSDL or service interface definition document. This property is optional. If not specified, the WSDL artifact category is defaulted to "wsdl".
    - *deploys-relationship-category*
      The name of the related asset category that the publisher will use in a service deployment asset to access the service interface definition asset. This property is optional. If not specified, the WSDL artifact category is defaulted to "deploys".
    - *schema-artifact-category*
      The name of the artifact category that the publisher will use for access to the XML schema documents. This property is optional. If not specified, the UDDIPublisher will not publish schemas into UDDI.
    - *targetNamespace-identifier-tmodel-key*
      UDDI key for the identifier tModel to be used with the "targetNamespace" identifier. This property is optional. If not specified, targetNamespace identifiers *will be associated with the uddi-org:general_keywords tModel.*
    - *data-locale*
      This property is used to specify the language to be associated with created UDDI entity descriptions as well as business service names. This property is optional. If not specified, the default language is set to "EN" (English).
- **Prerequisites:**
  The asset must have a valid WSDL artifact that is either a Service Interface Definition or a complete WSDL document, or a valid XML schema artifact in the case where the *schema-artifact-category* property has been specified.
- **Return Codes:**
    - 0 – success

Using Classification Criteria Sets as Filters

The Lifecycle Manager UDDI Governance Module takes advantage of another Lifecycle Manager feature: the ability for Lifecycle Manager users with the Usage Controller role to create groupings of classifier key/value pairs called Classification Criteria Sets. The ability to specify a Classification Criteria Set for use as an asset filter on a UDDIPublisher facilitates a number of possible UDDI publishing scenarios. Here are a couple of examples:

- **Deploying a Web service asset to UDDI based on its lifecycle state**
  A UDDIPublisher is configured using an asset-filter that specifies a certain value or

values for the "asset-state" classifier. In this scenario, assets of type "Service" or "Service Implementation" will not be published into UDDI until the assets have reached the specified state (set during asset edit) and are published or republished into Lifecycle Manager.

- **Publishing into multiple UDDI Registries**
  Multiple UDDIPublishers may be configured to publish different types of Web service assets into different registries based on the classification of the asset. By combining this approach combined with the lifecycle state concept described above, it is possible to publish Web services into different UDDI registries based on their lifecycle state. For example Web services may be published into a test UDDI registry early in their lifecycle and later published into a production UDDI registry when they reach a production-ready state.

Since UDDIPublishers are configured using SmartControls, customers can take full advantage of SmartControl filters, allowing UDDIPublishing to be customized not only by asset-filters (which may include asset-type) but also by owning Group, User, triggering Event, etc.

Configuring Lifecycle Manager Asset Types to enable UDDI Publication

As mentioned above, Lifecycle Manager comes preconfigured with two asset types that are automatically processed by its UDDI Governace Module: "Service" and "Service Implementation". These asset types, and more specifically the Capture Templates associated with these asset types (and of course the underlying Global Definition Template settings to support these Capture Templates), are defined with specific Classifier, Artifact, and Relationship types that are interpreted and processed by the Governance Module during the UDDI Publication process. Lifecycle Manager Library Administrators, Usage Controllers, and Asset Publishers may choose to extend these default definitions as appropriate to support their organization's desired software development asset lifecycle governance processes[67].

Populating Category and Identifier bags

Category and Identifier information is determined from classifiers on the Lifecycle Manager asset.
A KeyedReference is created from each Lifecycle Manager classifier/value pair. Additionally, an external mapping for UDDI may be defined for a classifier definition in the Lifecycle Manager global definition template that allows the publisher to associate the KeyedReference with a particular categorization tModel in the UDDI registry. Furthermore, if an external UDDI mapping is defined for a classifier value in the Lifecycle Manager asset definition template, that value will be used to select the appropriate value within the associated UDDI category.

[67] When configuring UDDIPublishers through SmartControls it is possible to allow UDDI publishing of other asset types that contain WSDL or schema artifacts. This is done by specifying these custom web service asset types in an asset-filter element and then specifying that asset-filter in the Filter element associated with the UDDIPublisher Listener.

Whether a KeyedReference is placed in a categoryBag or identifierBag of a UDDI entity is based on the prefix of the classifier's URI attribute. If no URI attribute is present, the KeyedReference is assumed to be a categorization and will be assigned to the uddi-org:general_keywords tModel.

Here is an example of a Lifecycle Manager classifier type that has been mapped onto a UDDI category. This example classifier is associated with a default tModel provided by the Microsoft UDDI Registry delivered as part of Windows 2003 Enterprise Server.

```
<define-classifier name="VisualStudio-WebService-Search-Categorization"
type="string" open="false" max-occurs="1"
<external-mapping key="UDDI"
      value="UDDI_CAT:uuid:4c1f2e1f-4b7c-44eb-9b87-6e7d80f82b3e"/>
help-text="Visual Studio Classification Scheme">
      <add-value value="Calendar">
            <external-mapping key="UDDI" value="0" />
      </add-value>
      <add-value value="Charting">
            <external-mapping key="UDDI" value="1" />
      </add-value>
      <add-value value="Collaboration">
            <external-mapping key="UDDI" value="2" />
      </add-value>
      <add-value value="Communication">
            <external-mapping key="UDDI" value="3" />
      </add-value>
      <add-value value="Data">
            <external-mapping key="UDDI" value="4" />
      </add-value>
      <add-value value="Dialup">
            <external-mapping key="UDDI" value="5" />
      </add-value>
      <add-value value="Encryption">
            <external-mapping key="UDDI" value="6" />
      </add-value>
      <add-value value="File">
            <external-mapping key="UDDI" value="7" />
      </add-value>
      <add-value value="Financial">
            <external-mapping key="UDDI" value="8" />
      </add-value>
      <add-value value="Graphics">
            <external-mapping key="UDDI" value="9" />
      </add-value>
      <add-value value="Help">
            <external-mapping key="UDDI" value="10" />
      </add-value>
      <add-value value="Imaging">
            <external-mapping key="UDDI" value="11" />
      </add-value>
      <add-value value="Mail">
            <external-mapping key="UDDI" value="12" />
      </add-value>
      <add-value value="Math">
            <external-mapping key="UDDI" value="13" />
      </add-value>
```

```
      <add-value value="Miscellaneous">
            <external-mapping key="UDDI" value="14" />
      </add-value>
      <add-value value="Printing">
            <external-mapping key="UDDI" value="15" />
      </add-value>
      <add-value value="Search">
            <external-mapping key="UDDI" value="16" />
      </add-value>
      <add-value value="Speech">
            <external-mapping key="UDDI" value="17" />
      </add-value>
      <add-value value="Weather">
            <external-mapping key="UDDI" value="18" />
      </add-value>
</define-classifier>
```

In this example the tModelKey for the UDDI tModel representing the "Visual Studio Search Categorization" category is entered in the "external-mapping" element of the classifier definition. Preceding the URI with "UDDI_CAT" indicates that the classifier is to be treated as a categorization ("UDDI_ID" would be used to indicate that the classifier is to be treated as an Identifier). Each value in the classifier in this example is mapped to a value in the specified UDDI category through the use of "external-mapping" elements.

UDDI Entities Created by the Lifecycle Manager UDDI Governance Module

Creating wsdlSpec tModels

tModels for Web service type assets are created as follows:

1) The asset name is used as the tModel name
2) The asset description is used as the tModel description
3) The tModel has a type of "wsdlSpec"
4) The WSDL artifact reference in the asset is set as the overviewDoc reference on the tModel[68].
5) The categoryBag and identifierBag on the tModel are populated from the asset classifiers (see "Populating Category and Identifier Bags" for details)
6) An identifier will be placed in the tModel's identifierBag that indicates the Lifecycle Manager asset ID[69].

Creating Business Services

Business services are created in UDDI as follows:

1) The service element's name attribute in the WSDL spec is used the business service name in the UDDI registry.
2) The Lifecycle Manager asset's description is used as the business service description
3) The categoryBag is populated from the asset classifiers (see "Populating Category and Identifier Bags" for details)

[68] The wsdl artifact may be either "by value" (stored within Lifecycle Manager) or "by reference" (referencing a document outside of Lifecycle Manager). In the former case the reference used within UDDI will be to the Lifecycle Manager artifact retrieval servlet.

[69] This identifier will reference a dedicated "logiclibrary-com:asset-id" tModel that is created in the UDDI registry when the UDDIPublisher is created

4) If the "businessKey" classifier is found in asset, that key is set as the businessKey in the UDDI business service, otherwise the default business key set for the UDDIPublisher is used.

Creating Binding Templates

Binding templates are created in UDDI as follows:

1) The asset description is used as the binding template description
2) The access point of the binding template is set from the location attribute within the port definition in the WSDL artifact.
3) A TModelInstanceInfo is created in the TModelInstanceDetails on the binding template that references the tModel associated with the service interface definition for this service[70].

Creating specification tModels

tModels for assets representing XML schemas are created as follows:

1. The asset name is used as the tModel name
2. The asset description is used as the tModel description
3. The tModel has a type of "specification"
4. The schema artifact reference in the asset is set as the overviewDoc reference on the tModel[71].
5. The categoryBag and identifierBag on the tModel are populated from the asset classifiers (see "Populating Category and Identifier Bags" for details)
6. An identifier is added to the tModel's identifierBag that indicates the Lifecycle Manager asset ID[72].
7. A "targetNamespace" identifier is added to the identifier bag with a value set to the schema's targetNamespace. This identifier is associated with the "uddi-org:general_keywords" tModel by default but may optionally be associated with a specified identifier tModel[73].

Importing UDDI Services

Lifecycle Manager provides a UDDIImporter to allow web service assets to be imported from UDDI. As with other Lifecycle Manager importers, the UDDIImporter is defined in the <importers> element of the Lifecycle Manager Process Configuration document. The configuration details for the UDDIImporter follow:

UDDIImporter

- **Behavior:**
  Creates Web service assets from BusinessServices in a UDDI registry.
- **Properties:**
  - *description*
    The description of the importer.
    Defaults to "Import services from UDDI". (optional)

[70] In the case of a complete web service asset, the reference will be to the tModel created specifically for this asset. In the case of a service implementation asset, the reference will be to the tModel associated with the service asset specified as in the "implements" relationship of the service implementation asset.

[71] The wsdl artifact may be either "by value" (stored within Lifecycle Manager) or "by reference" (referencing a document outside of Lifecycle Manager). In the former case the reference used within UDDI will be to the Lifecycle Manager artifact retrieval servlet.

[72] This identifier will reference a dedicated "logiclibrary-com:asset-id" tModel that is created in the UDDI registry when the UDDIPublisher is created

[73] This is done by specifying the UUID of the desired identifier tModel in the "targetNamespace-identifier-tmodel-key" property of the UDDIPublisher.

- *inquiry-url*
  Inquiry URL for the UDDI registry (e.g. “http://westlake/uddi/inquire.asmx”)
  This property is mandatory.
- *registry-user*
  The user id to use when publishing to the UDDI registry. All entities created in the UDDI registry will be owned by this user.
  This property is mandatory. This property is mandatory.
- *registry-password*
  The password to use when publishing to the UDDI registry.
  This property is mandatory.
- *registry-alias*
  A unique name for the source UDDI registry that may be used to disambiguate properties when multiple registries are associated with this Lifecycle Manager instance (e.g “ProductionUDDI”, “TestUDDI”). This property is mandatory.
- *wsdl-category*
  The name of the artifact category that the publisher will use for access to the WSDL or service interface definition document. This property is optional. If not specified, the WSDL artifact category is defaulted to “wsdl”.
- *implements-relationship-category*
  The name of the related asset category that the publisher will use in a service implementation asset to access the service interface definition asset. This property is optional. If not specified, the WSDL artifact category is defaulted to “deploys”.
- *wsdl-artifact-containment*
  Determines the containment of the wsdl artifact. Valid values are “by-value” and “by-reference”. This setting is only meaningful in the case where the artifact is uploaded from a URL; loading a WSDL document from a file will force artifact containment to be “by-value”.
  Defaults to “by-reference”. (optional)
- *service-asset-version*
  The asset version that will be used for assets created by this importer. Defaults to “1.0” (optional).
- *service-interface-asset-type*
  The asset-type used for service assets created by this importer.
  Defaults to “Web service” (optional).
- *service-implementation-asset-type*
  The asset-type used for service implementation assets created by this importer.
  Defaults to “Web service Deployment” (optional).
- *service–interface-template*
  The capture template to use for service assets created by this importer.
  Defaults to “WebService” (optional).
- *service–implementation-template*
  The capture template to use for service implementation assets created by this importer.
  Defaults to “WebService Deployment” (optional).

- *create-note*
  Note used for creation of schema Assets.
  Defaults to "created by UDDIImporter" (optional).
- *submit-note*
  Note used for submission of schema Assets.
  Defaults to "submitted by UDDIImporter" (optional).
- *max-results*
  The maximum number of services that will be returned from a query. Defaults to 100 (optional).
- *security-url*
  Security URL for the associated UDDI registry (optional).

- **Resulting Assets**
  A service interface asset of type specified by the importer properties will be created to represent the BusinessService. A service implementation asset of type specified by the importer properties will be created for each child bindingTemplate. Service implementation assets will have the source WSDL document as an artifact and an implementation relationship of type specified by the importer properties to the parent businessService asset. Created assets will have classifiers set according to the external mappings defined in the Global Definition Template as described in the "Populating category and identifier bags" section earlier in this document. Note that only those classifications and identifiers for which there is an explicit mapping defined in the GDT will be populated as classifiers on the created assets.
  Additionally, service interface assets and service implementation assets will have "businessServiceKey" and "bindingTemplateKey" properties set respectively.

Advanced Topics

Overriding UDDI Entity Creation

It is possible for the provider of an asset to specify that an existing tModel or businessService in the UDDI registry is to be used rather than a tModel or businessService created specifically for the asset.

tModel creation can be overridden by specifying the key of an existing wsdlSpec tModel in the "wsdlTModelKey" classifier of a web service type asset.

BusinessService creation can be overridden by specifying the key of an existing businessService in the "businessServiceKey" classifier of service and service implementation type assets. Additionally, the owning business entity can be overridden through use of the "businessKey" classifier.

Handling Multiple Services and Ports

The Lifecycle Manager UDDIPublisher will allow multiple services and ports to be defined within the WSDL artifact of a single complete web service or web service deployment asset.

Publishing of such an asset will result in multiple businessService and bindingTemplate entities being created in the UDDI registry.

Although this scenario is allowed by the UDDIPublisher, LogicLibrary recommends that complete web service and web service deployment assets be limited to a single service and port definition as this is the recommended OASIS best practice and is the most intuitive and practical unit of deployment and maintenance.

Appendix V: Customizing Tasks in Configuration Designer

The tasks shown on the palette of the Configuration Designer are defined in the tasktemplates.xml file. The tasktemplates.xml file can be found in the Document-Source folder of a configuration designer project. If not found, disable the Configuration Designer filter which will expose the tasktemplates.xml and other files typically not of interested to users.

A task definition associates a listener with a task, defining return code behavior, and managing the properties of the listener. The Library Configuration Guide documents listeners provided by SOA in Appendix A. In the example below, a GenericRequestHandler listener will be inserted into the workflow when the Notify Submitter task is dropped on the canvas.

```
<task-template category="basic" desc="Sends an e-mail to the original
   submitter of the request" name="Notify Submitter" ordinal="3" type="GRH">
   <listeners>
      <listener class="GenericRequestHandler" name="NotifySubmitter">
         <return-codes>
            <return-code action="WARN" event="WARNING_LISTENER_FAILED"
               value="-1"/>
         </return-codes>
         <parameters>
            <property default-value="NOTIFY_SUBMITTER_APPROVAL_REQUIRED"
               desc="The identifier of the e-mail message to send" display-
               name="Message Id" name="submitter-message-id" ordinal="1"
               required="true" type="MessageId"/>
         </parameters>
      </listener>
   </listeners>
</task-template>
```

The return codes of the listener are mapped to actions which typically raise an existing error or warning event.

Properties of the listener can be exposed to the user via the property element. In the example above, the user will be offered a “Message Id” property in the Properties View when the task is selected. This will set the submitter-message-id property for the GenericRequestHandler. The ‘type’ attribute of the property will offer a selection list of files found in the Document-Source/messages directory.

Properties values can be hardcoded in the template via a preset. Presets will not be exposed in the Properties View. The preset values are applied when the task instance is created (i.e. the task

is dropped onto the designer canvas) and any time the lpc file is reopened. This offers a convenient means to update properties pointing to external systems when those system locations change.

Please refer to the tasktemplates.xsd for additional guidance. If needed, contact Support or Professional Services for assistance customizing the tasktemplates.xml.

Appendix W: Community Manager Integration

Lifecycle Manager may be configured to capture and publish assets representing APIs into the Community Manager environment. This appendix describes the configuration steps necessary to accomplish this.

LPC Configuration

Asset Types

There are two asset-types that should be defined to represent an API and its optional proxy endpoints. These are the types "API" and "API Proxy Endpoint" respectively.
Here is the standard definition for the API type:

```
<asset-type-definition name="API">
 <templates>
   <template name="API - Specified" />
 </templates>
 <edit-roles>
   <role name="Service Developer" />
 </edit-roles>
</asset-type-definition>
```

As shown, templates and roles may be assigned to the type.

Next is the API Proxy Endpoint definition, note that this is a subsidiary asset type:

```
<asset-type-definition name="API Proxy Endpoint" subsidiary = "true">
  <templates>
      <template name="API Proxy Endpoint - General"></template>
  </templates>
 <edit-roles>
   <role name="Service Developer" />
 </edit-roles>
 <prototypes>
   <prototype name="Production Endpoint">
     <external-mappings>
       <external-mapping owner="APIValidator" key="Production" />
     </external-mappings>
     <template>API Proxy Endpoint - General</template>
     <default-version>1</default-version>
     <default-description>Production API endpoint</default-description>
     <default-overview>Production environment proxy endpoint for the API</default-overview>
   </prototype>
   <prototype name="Sandbox Endpoint">
     <external-mappings>
       <external-mapping owner="APIValidator" key="Sandbox" />
     </external-mappings>
     <template>API Proxy Endpoint - General</template>
     <default-version>1</default-version>
     <default-description>Sandbox API endpoint</default-description>
     <default-overview>Sandbox environment proxy endpoint for the API</default-overview>
   </prototype>
 </prototypes>
```

```
    </asset-type-definition>
```

In addition to the subsidiary designation the type declares two “prototypes”. These are used to enable the auto-creation of proxy endpoint assets when the “generate proxy” option is chosen for the parent API asset.

Finally, a Legal Agreement type should be declared to represent legal agreements to be exposed to users of an API:

```
    <asset-type-definition name="Legal Agreement">
      <templates>
        <template name="Legal Agreement - General" />
      </templates>
      <edit-roles>
        <role name="Business Analyst" />
      </edit-roles>
    </asset-type-definition>
```

Federated System

A federated-system should be declared to represent the connection to the CommunityManager instance:

```
     <federated-system name="Community Manager"
class="com.logiclibrary.integrations.cm.CommunityManager">
      <properties>
        <property name="user" value="someUser" />
        <property name="password" value="thePassword" encrypt="true" />
       <property name="endpoint" value="http://MyCMServer:9900" />
      </properties>
    </federated-system>
```

The connection properties for CoummunityManager are as follows:

- **user**
  The user name to use when connecting to the CommunityManager instance (required)
- **password**
  The designated user’s password (required)
- **endpoint**
  The URL (host and port) of the CommunityManager instance (required)
- **context-root**
  The context root of the CommunityManager application. This is an optional property and defaults to “cm”.
- **validate-connection**
  Boolean property indicating that the connection to CommunityManger should be validated when the LPC is uploaded. This is an optional property and defaults to “true”.

Additionally, the CommunityManager federated-system allows for a number of properties that can optionally be used to assign non-standard names to the classifiers, relationships and artifacts that the integration depends on. Use of these properties is only necessary if the default element names are not used. These properties and their default values are shown in the following table:

Property Name	***Default Value***
api-asset-type	API
production-endpoint-classifier-name	production-endpoint
sandbox-endpoint-classifier-name	sandbox-endpoint
proxy-endpoint-asset-type	API Proxy Endpoint

generate-proxy-classifier-name	generate-proxy
tags-classifier-name	keyword
auto-approve-classifier-name	consumption-auto-approved
cname-classifier-name	cname
deployment-zone-classifier-name	deployment-zone
url-path-classifier-name	url-path
document-path-classifier-name	document-path
document-display-name-classifier-name	document-display-name
document-description-classifier-name	document-description
visibility-classifier-name	accessibility
content-length-header-classifier-name	content-length-header
http-10-classifier-name	http-10
endpoint-type-classifier-name	endpoint-type
policies-classifier-name	proxy-policies
target-rest-definition-category	target-rest-definition
proxy-rest-definition-category	proxy-rest-definition
avatar-artifact-category	avatar
agreement-artifact-category	documentation
oauth-artifact-category	rest-deployment-oauth-definition
production-proxy-endpoint-relationship	production-proxy-endpoint
sandbox-proxy-endpoint-relationship	sandbox-proxy-endpoint
agreement-relationship	enforces

Content Handlers

CommunityManager integration relies on some special UI wizards for capturing operation and OAuth information for an API. This is accomplished through the use of content-handlers defined as follows:

```
<artifact-content-handler name="RESTTarget" class="RESTContentHandler">
    <properties>
        <property name="proxy-mode" value="false" />
    </properties>
</artifact-content-handler>
<artifact-content-handler name="RESTProxy" class="RESTContentHandler">
        <properties>
        <property name="proxy-mode" value="true" />
    </properties>
</artifact-content-handler>
<artifact-content-handler name="OAuthDetails" class="OAuthContentHandler">
     <properties>
        <property name="federated-system-name" value="Community Manager" />
    </properties>
</artifact-content-handler>
```

The first of these, “RESTTarget”, is used to enable the operations wizard for the target operations artfact. “RESTProxy” enables the wizard in a constrained mode for use with the proxy-specific operations artifact. Finally, OAuthDetails is used for the OAuth configuration artifact. Note that the CommunityManager federated-system must be specified as a property for the OAuthContentHandler.

Artifact Source

CommunityManager integration introduces a single new artifact-source for use in accessing an API's detail page:

```
<artifact-source name="CM" class="CommunityManagerArtifactSource">
  <properties>
    <property name="federated-system-name" value="Community Manager" />
  </properties>
</artifact-source>
```

Value Sources

A value-source is defined to allow for the retrieval of deployment zones from the CommunityManager instance:

```
<value-source name="DeploymentZones" class="DeploymentZoneValueSource">
   <properties>
     <property name="federated-system-name" value="Community Manager" />
    </properties>
</value-source>
```

Another is defined for retrieving policies:

```
<value-source name="CMPolicies" class="CMPolicyValueSource">
  <properties>
    <property name="federated-system-name" value="Community Manager" />
   </properties>
</value-source>
```

Asset Validators

Since API assets contain proxy endpoints as subsidiary assets the SubsidiaryAssetValidator must be defined:

```
<asset-validator name="SubsidiaryAssetValidator" class="SubsidiaryAssetValidator" >
      <validation-context>ASSET_CREATION</validation-context>
      <validation-context>ASSET_UPDATE</validation-context>
      <validation-context>ASSET_NEW_VERSION_CREATION</validation-context>
      <validation-context>ASSET_LIKE_CREATION</validation-context>
</asset-validator>
```

Additionally, an APIValidator must be defined to run against API assets:

```
<asset-validator name="APIValidator" class="APIValidator">
   <validation-context>ASSET_CREATION</validation-context>
   <validation-context>ASSET_UPDATE</validation-context>
   <validation-context>ASSET_PARTIAL_UPDATE</validation-context>
  <asset-filter-name>asset-type:API</asset-filter-name>
</asset-validator>
```

Listener

The publication of an API asset into Community Manager is accomplished via the CommunityManagerPublisher listener. This listener is normally configured as a task in Configuration Designer. The optional property "federated-system-name" is used to set the name of the CommunityManager federated-system. If not specified, the federated-system name is assumed to be "Community Manager".

Templates

GDT

The following integration specific classifiers should be added to the GDT:

```
   <!-- API -->
  <define-classifier display-name="API Type" help-text="Type of API" max-occurs="1" name="api-
type" open="false" type="string" value-ordering="GDT">
    <add-value value="REST"/>
  </define-classifier>
  <define-classifier display-name="Production Endpoint" help-text="API Physical Production
Endpoint" max-occurs="1" name="production-endpoint" type="string"/>
  <define-classifier display-name="Sandbox Endpoint" help-text="API Physical Sandbox Endpoint"
max-occurs="1" name="sandbox-endpoint" type="string"/>
  <define-classifier display-name="Generate Proxy?" help-text="Is a proxy for this endpoint to be
automatically generated?" max-occurs="1" name="generate-proxy" type="boolean"/>
  <define-classifier display-name="Add Content-Length Header" help-text="Set to 'true' if you
need to send the Content-Length heading to the target API. This disables chunked encoding." max-
occurs="1" name="content-length-header" type="boolean"/>
  <define-classifier display-name="Use HTTP 1.0" help-text="Set to 'true' if you need to force
the HTTP version to 1.0 for the target API. This is unlikely to be needed." max-occurs="1"
name="http-10" type="boolean"/>
<define-classifier display-name="Proxy Policies" help-text="Select the policies that you would
like the system to enforce on the proxy. You should select one security policy (e.g Simple Header
Security) and a monitoring policy if you want to see charts and logs." name="proxy-policies"
type="string" value-source = "CMPolicies"/>

  <!-- Proxy Endpoint -->
  <define-classifier display-name="Proxy Endpoint Type" help-text="Type of the proxy endpoint
e.g. production or sandbox" max-occurs="1" name="endpoint-type" type="string">
    <add-value value="Production"/>
    <add-value value="Sandbox"/>
  </define-classifier>
  <define-classifier display-name="Deployment Zone" help-text="Target environment for this
deployment (applicable when generating proxy)" max-occurs="1" name="deployment-zone"
type="string" value-source="DeploymentZones"/>
  <define-classifier display-name="Consumption auto-approved?" help-text="Is app consumption of
this API to be automatically approved?" max-occurs="1" name="consumption-auto-approved"
type="boolean"/>
  <define-classifier display-name="CNAME" help-text="Canonical name record e.g. 'api.company.com'
(applicable when generating proxy)" max-occurs="1" name="cname" type="string"/>
  <define-classifier display-name="Path" help-text="Qualifying path for published URL e.g.
'api.company.com/path' (applicable when generating proxy)" max-occurs="1" name="url-path"
type="string"/>

  <!-- Legal Agreement -->
  <define-classifier display-name="Document Name" help-text="Published display name for the legal
agreement document" max-occurs="1" name="document-display-name" type="string"/>
  <define-classifier display-name="Document Description" help-text="Published description for the
legal agreement document" max-occurs="1" name="document-description" type="string"/>
```

Note the use of the “DeploymentZones” value-source for the deployment-zone classifier and “CMPolicies” value-source for the proxy-policies classifier.

The following API-specific artifacts are defined:

```
<define-artifact category="avatar" containment="by-value" display-name="Avatar Image" help-
text="75x75 image to represent API in community access environment" max-occurs="1"/>
<define-artifact category="target-rest-definition" containment="by-value" content-
handler="RESTTarget" display-name="Target REST API Definition" help-text="Operations for the
target REST API" max-occurs="1" queryable="true"/>
<define-artifact category="proxy-rest-definition" containment="by-value" content-
handler="RESTProxy" display-name="Proxy REST API Definition" help-text="Operations for the proxy
REST API" max-occurs="1" queryable="true"/>
<define-artifact category="rest-deployment-oauth-definition" containment="by-value" display-
name="REST Deployment OAuth Definition" content-handler = "OAuthDetails" help-text="Definition of
OAuth parameters associated with the API deployment" max-occurs="1" queryable="true"/>
```

Note the use of content-handlers for the REST and OAuth definitions.

Document artifacts within the API asset to be published to CM should be marked using the external-mappings:

- *cm-publish*
  Indicates the artifact should be published to CM
- *cm-default-document*
  Indicates the artifact is the default document for the API in CM
- *cm-toc-document*
  Indicates that the document should appear in the API's table of contents

Here is an example declaration:

```
 <define-artifact category="usage-guide" display-name="Usage Documentation" help-text="Documentation
describing how to use the asset">
    <external-mapping key = "cm-publish" value="true"/>
    <external-mapping key = "cm-default-document" value="true"/>
    <external-mapping key = "cm-toc-document" value="true"/>
  </define-artifact>
```

Note that artifacts to be published to CM (including legal documents should be constrained to "by-value" containment in the templates.

Finally, these API-specific relationships are defined:

```
<define-asset-relationship disable-manual-entry="true" disable-search="true" display-
name="Production Proxy Endpoint" help-text="The proxy production endpoint of this API" max-
occurs="1" name="production-proxy-endpoint" reverse-role-name="Production proxy endpoint of"
target-asset-stereotype="asset-type:API Proxy Endpoint"/>
<define-asset-relationship disable-manual-entry="true" disable-search="true" display-
name="Sandbox Proxy Endpoint" help-text="The proxy sandbox endpoint of this API" max-occurs="1"
name="sandbox-proxy-endpoint" reverse-role-name="Sandbox proxy endpoint of" target-asset-
stereotype="asset-type:API Proxy Endpoint"/>
<define-asset-relationship disable-manual-entry="true" display-name="Enforces" help-text="Legal
agreement enforced for this API" name="enforces" reverse-role-name="Enforced By"/>
```

Capture Templates

Capture templates are introduced for the types API, API Proxy Endpoint, and Legal Agreement using the elements defined above. These templates are available in the Demo Configuration available from SOA Support and will not be reproduced here.

Appendix X: Rest Integration Interface

In addition to a Java client using SOAP, the Lifecycle Manager/Portfolio Manager products also offer a simple generic REST API for invoking scripted "functions" that run on the LM server and may invoke local extension APIs.

Defining Functions:

Functions are defined in the LPC within the <functions> element as in the following example:

```
<function name="GetAssetStatus" class="ScriptFunction" authentication="BASIC" >
     <properties>
            <property name="script-id" value="scripts/GetAssetStatus.bsh" />
     </properties>
 </function>
```

Functions are required to have a unique name, are generally implemented using the "ScriptFunction" class and may be provided with configuration properties. In the case of a scripted function, the path to the script in the document source must be provided in the "script-id" property.
The "authentication" attribute allows a function provider to choose between cookie-based ("COOKIE"), HTTP Basic ("BASIC") or no authentication at all ("NONE")[74] when the function is called. The default authentication is "BASIC"[75].

A scripted function logically implements the following signature:

```
void execute(Context context,
             com.soa.repositorymanager.external.FunctionInput input,
             com.soa.repositorymanager.external.FunctionOutput output,
             java.util.Properties configurationProperties) {
```

Where:

- *context* is an IN parameter that contains the libraryId and userId
- *input* is an IN parameter that contains information from the REST request. This includes query parameters stored as properties as well as a getMethodInput() property to access the body of the REST method (normally in JSON format). In the case of a multi-part request, the input object may contain a Document holding file contents.
- *output* is an IN_OUT parameter to store information for the method response. The class includes the method setResponseCode() for setting the http response code as well as a setMethodOutput() method for setting the body of the response (normally in JSON format). The FunctionOutput object may also contain a Document holding file contents.
- *configurationProperties* is an IN parameter that contains the properties set in the definition of the Function in the LPC.

Invoking Functions with REST

REST-style functions are invoked using URLs of the following format:
http://<lm-server>/<context-root>/servlet/rest/integration/<libraryname >/<functionname ><parameters>
For example:
http://myServer/lm/servlet/rest/integration/MyLibrary/GetAssetStatus?name=Service&version=1.0

Clients invoking Functions using COOKIE authentication must first login using the following URL:
http:// <lm-server>/<context-root>/servlet/rest/login/<libraryname>
Passing the JSON structure:
{"user":"<user id>", "password":"<password>"}
The response to this call will have a "Set-Cookie" header containing the authentication cookie. This cookie needs to be set in the "Cookie" header of requests to the function.

[74] Note that if "NONE" is chosen for authentication, the context provided to the function script will not contain a userId
[75] If "BASIC" authentication is used with Weblogic Application Servers, the line <enforce-valid-basic-auth-credentials>false</enforce-valid-basic-auth-credentials> must be added to the < security-configuration> element in the server's config.xml. If this flag is not set to false, the server itself will issue challenges on requests with HTTP basic authentication.

Appendix Y: WSDL / Schema resolution

Various listeners and validators need to build up a full view of the service. This is done by attempting to resolve WSDL imports and schema imports and includes to other artifacts on service or schema assets, or externally to other resources hosted on web servers.

WSDL/Schema resolution algorithm

To build up the complete WSDL/Schema, we need to resolve imports and includes. This is done using the following algorithm. Behavior can be modified using the properties below.

1. If the schema or WSDL contains an import (which should have a namespace and location), we search for assets containing that namespace using the namespace classifer.
    a. We potentially get back a list of assets containing the namespace classifier, we eliminate any assets that don't have wsdl/schema artifacts, or if the resource is itself, or if the filename in the location classifier (not the entire location classifier which also has path information) is not the same as the filename of the resource being imported.
    b. If there is only one asset found that matches the above, we've identified the resource and continue processing other imports / includes
2. If the schema or WSDL contains an import of include that refers to a location, we do the following.
    a. If the resource being included is an absolute URI (one with a protocol e.g. http://soa.com/wsdl/a.wsdl), we search for assets that contain a location classifier with that URI. If we find one resource, we stop looking and use it.
    b. If the resource being included is a relative URI, we resolve that URI based on the location classifier of the including asset and search for assets with a location classifier with the combined URI. For example, if asset A with location classifier foo/wsdl/foo.wsdl includes a schema with a location of ../xsd/foo.xsd, we search for assets with a location classifier or foo/xsd/foo.xsd. If only one resource is found, we stop looking.
    c. If the above fails, we simply try to look for assets with location classifier of the import/include's location. So in the above example, we'd search for assets with a location classifier of ../xsd/foo.xsd. If only one asset is found, stop looking
3. If by this time we still haven't found a matching resource, we fall back to simply using the namespace as a URL. So if the import is for namespace="http://soa.com/xsd/foo.xsd", even though the spec doesn't say that has to resolve to a valid resource, we still try to see if it returns one.
4. If none of the above matching techniques worked, we fail with resource could not be found.

Installation Properties

There are three installation properties that can be used to modify the above behavior for the install. These are set with the SetInstallationProperty administration command

- *wsdl.resolution.priority*
  This can be set to "namespace" or "location". This controls whether the namespace resolution (step 1 above) takes place before location resolution (step

2), or vice-versa. The default behavior is "namespace", which means resolution occurs in the order specified above. If set to "location", then the behavior is that location resolution (step 2 above) takes place before namespace (step 1).

- *wsdl.resolution.skip.namespace*
  This can be set to "true" or "false". This controls whether the namespace resolution (step 1 above) takes place. Default is "false".
- *wsdl.resolution.skip.location*
  This can be set to "true" or "false". This controls whether the location resolution (step 2 above) takes place. Default is "false".

LPC Properties

These properties are specified in the LPC on certain tasks (i.e. WSDLValidator) to control the behavior of where it expects to find this information for each asset.

WSDL(Service) properties

- *service-artifact-category*
  This is the artifact category containing the service (WSDL). The default is "message-definition"
- *packed-service-artifact-category*
  This is the artifact category containing a packed service (ZIP). The service and relevant schemas can be packaged together in this artifact. The default is "packed-service"
- *service-namespace-classifier*
  The namespace classifier that contains the target namespace of the service. This is used to resolve services that are located within Lifecycle Manager. The default is "target-namespace".
- *service-name-classifier*
  The name of the classifier that contains the service name that the asset represents. Multiple services within a WSDL will cause errors if this classifier is not set. The default is "service-name".
- *service-location-classifier*
  The name of the classifier that contains the location for the WSDL/service. This value is used to locate resources resolved relative to the including resource, and also for absolute references. The default is "location".

WSDL(service) and schema properties

- *schema-artifact-category*
  This is the artifact category containing schemas. The default is "schema-definition"
- *schema-namespace-classifier*
  The namespace classifier that contains the target namespace of the schema. This is used to resolve schemas that are located within Lifecycle Manager. The default is "target-namespace".
- *schema-location-classifier*
  The name of the classifier that contains the location for the WSDL/service. This value is used to locate resources resolved relative to the including resource, and also for absolute references. The default is "location".