Policy Manager and Lifecycle Manager

Integration Guide

Policy Manager and Lifecycle Manager

Integration Guide
Version: 7.0
July, 2015

Copyright

Trademarks

Akana, Inc. (formerly SOA Software, Inc.)

Akana, Inc.
12100 Wilshire Blvd, Suite 1800
Los Angeles, CA 90025
(866) SOA-9876
www.akana.com
info@akana.com

Disclaimer

Contents

Overview

The Smart Controls framework allows the Repository/Lifecycle Manager (RM) product to be tightly integrated with the Akana Policy Manager (PM) product allowing for end-to-end-governance of the service lifecycle. Policy Manager integration is facilitated primarily with listeners and related elements defined in the Library Configuration document. While this document does not describe the actual service flows and associated use cases, it does describe the integration points between the products and the associated library configuration elements.

Installation of Policy Manager 5.2 (EJB) EAR

There are certain components of Repository/Lifecycle Manager that communicate with Policy Manager 5.2 using an EJB that is deployed as an EAR file. This Policy Manager EJB allows exposes certain APIs that are not available through the normal Policy Manager interaction. The following instructions should be followed to install the EAR into the application server. Note that the Policy Manager EJB EAR only supports one Policy Manager system per installation. These instructions should be followed if you are configuring a system to communicate with Policy Manager 5.2. If you are using Policy Manager 6.0, see the next section.

Follow these instructions to generate and install the EAR.

1. Copy the *PM_HOME*/sm52/config/bootstrap.properties and dems.properties file of the Policy Manager system you are communicating with to the *RM_HOME*/conf directory
2. Copy any Service Manager installed updates from the Policy Manager server in *PM_HOME*/ installed_updates/service_manager/lib directory to the *RM_HOME*/deploy/app/policymanager52-updates directory. E.g.:
    - mkdir /opt/rm_application/deploy/app/policymanager52-updates
    - cp /opt/soa_sw/sm52/installed_updates/service_manager/lib/* \ /opt/rm_application/deploy/app/policymanager52-updates
3. Run *RM_HOME*/bin/install build-policymanager-ear (The ear will be created in *RM_HOME*/deploy/policymanager.ear
4. Deploy the ear according to the application server you are using (paths may need to be adjusted accordingly)

JBoss

- Create a directory in *JBOSS_HOME*/server/default/deploy/policymanager.ear
- Unjar the ear file in that directory

- Edit *JBOSS_HOME*/server/*server_name*/conf/jboss-service.xml
    - Locate the NamingService mbean XML section and modify “CallByValue” from false to true
    - It should resemble: <attribute name="CallByValue">true</attribute>
- Edit *JBOSS_HOME*/server/*server_name*/deploy/ear-deployer.xml
    - Locate the EARDeployer mbean XML section and modify “CallByValue” from false to true
    - It should resemble: <attribute name="CallByValue">true</attribute>

WebSphere

- Using the WebSphere admin console, install the new ear file.
- Ensure the EAR file is deployed to the same application server as RM is installed on.

WebLogic

- Create a directory next to the RM deployed app directory, for example, /opt/bea/user_projects/domains/base_domain/applications/policymanager
- Unjar the ear in that directory
- Login to the WebLogic console and step through deploying a new ear file, pointing it at the directory you created above.

Installation of Policy Manager 6.0

The integration with 6.0 uses web service APIs, so there is no need to install an EAR as there was for Policy Manager 5.2 integration. However, the integration however does require installation of these WebService APIs. Follow these steps to install the WebService APIs.

1. Install Policy Manager 6.0 according to the Policy Manager installation instructions
2. Install the Integration and Workflow Feature Option Pack
    a. Download the com.soa.integration.services zip file from the Akana support site under Downloads -> PolicyManager -> PM 60 -> RepositoryManagerIntegration directory
    b. Extract the file to a directory on the Policy Manager server
    c. On the repository tab of the Policy Manager 6.0 Admin console, add a new URL to the “repository.xml” file extracted above. E.g. file:/C:/temp/pm-rm/repository.xml

d. Go to the Available features and install the "SOA Software Integration and Workflow Services feature" -- you should have already installed the Policy Manager Console and Services feature

e. You may need to restart Policy Manager after installing the Integration Option Pack

3. Due to Web Service Security, the clock times on the Repository/Portfolio Manager system and Policy Manager system should be synchronized to a time source (e.g. using NTP, or a domain server).

4. The PKI Certificate from Policy Manager 6.0 will need to be exported and imported into Repository/Lifecycle Manager or Portfolio Manager. This step only needs to be completed in instances where access to Policy Manager is secured and restricted to HTTPS.

    a. Login to Policy Manager 6.0, expand the "SOA Software Policy Manager" organization, expand the Services item, and select the "Repository Integration Manager Service". On the right, select the "Manage PKI Keys" link.

    b. Export the X.509 certificate and save it.

    c. See the next section "**Error! Reference source not found.**" for instructions on importing it into your library.

Configuring Policy Manager as a Federated System

Connections to external registries such as Policy Manager are facilitated by specifying a specific <federated-system> element in the <federated-systems> section of the Library Configuration document. A <federated-system> element represents not only the connection to an external system but the identity of the system. Any data stored in RepositoryManager for that system will be scoped by the name of the federated-system. For this reason federated-system names must be unique and cannot be changed. The federated-system class for Policy Manager is PolicyManager. Depending on if you are integrating with Policy Manager 5.2 or 6.0, refer to the correct Policy Manager federated system details below.

Policy Manager Details for Policy Manager 5.2

- **Purpose:**
  Represents a Policy Manager installation

- **Class:** com.logiclibrary.registry.PolicyManager

- **Properties:**

    - *registry-host*
      The root URL to the Policy Manager installation.

Example: “http://myPolicyManagerServer.mycompany.com”
(Required)

- *user*[1]
  User for authentication with Policy Manager
  (Required)

- *password*
  Password for authentication with Policy Manager
  (Required)

- *data-locale*
  Locale associated with data (such as name and description) in Policy Manager. This should be a UDDI compliant locale.
  This property is optional. If not specified, a value of “en” is assumed.

- *inquiry-url*
  URL of the Policy Manager UDDI V2 Inquiry service. This property is optional and should only be specified if the service is not at the standard path relative to the registry-host URL. Default is “<registry-host>:9901/uddi/inquiry_v2”

- *publish-url*
  URL of the Policy Manager UDDI V2 Publish service. This property is optional and should only be specified if the service is not at the standard path relative to the registry-host URL. Default is “<registry-host>:9901/uddi/publish_v2”

- *technote-service-url*
  URL of the Policy Manager TechNote service. This property is optional and should only be specified if the service is not at the standard path relative to the registry-host URL. Default is “<registry-host>:9901/wsdltechnote”

- *policy-service-url*
  URL of the Policy Manager ServiceLevel Policy service. This property is optional and should only be specified if the service is not at the standard path relative to the registry-host URL. Default is “<registry-host>:9904/servicelevel”

- *contract-service-url*
  URL of the Policy Manager Contracts service. This property is optional and should only be specified if the service is not at the standard path relative to the registry-host URL. Default is “<registry-host>:9904/contract”

[1] The user specified should have admin privileges in Policy Manager.

- *compliance-service-url*
  URL of the Policy Manager Compliance Policy service. This property is optional and should only be specified if the service is not at the standard path relative to the registry-host URL. Default is “<registry-host>:9904/compliance”

- *workflow-service-url*
  URL of the Policy Manager Workflow service. This property is optional and should only be specified if the service is not at the standard path relative to the registry-host URL. Default is “<registry-host>:9904/workflow”

- *application-url*
  URL of the Policy Manager application, used in forming URLs to service and contract detail pages. This property is optional and should only be specified if the application is not at the standard path relative to the registry-host URL. Default is “<registry-host>:9900”

- *rest-url*
  URL of the Policy Manager REST servlet. This property is optional and should only be specified if the servlet is not at the standard path relative to the registry-host URL. Default is “<registry-host>:9900/rest/services/”

Example Configuration

```
<federated-systems>
  <federated-system name="SMTest" class="PolicyManager">
    <properties>
      <property name="registry-host" value="http://PolicyManagerServer.com"/>
       <property name="user" value="Administrator"/>
       <property name="password" value="password" encrypt="true"/>
     </properties>
  </federated-system>
</federated-systems>
```

Policy Manager Details for Policy Manager 6.0

- **Purpose:**
  Represents a Policy Manager installation.

- **Class:** com.logiclibrary.registry.PolicyManager

- **Properties:**

  - *registry-host*
    The root URL to the Policy Manager installation.
    Example: “http://example.com:9900”
    (Required)

- *user*[2]
  User for authentication with Policy Manager
  (Required)

- *password*
  Password for authentication with Policy Manager
  (Required)

- *version*
  Should be "6.0.x" depending on what version of Policy Manager you are using. If you see errors contacting the workflow service it could indicate that this property is set incorrectly. If it is not set, it assumes the registry host is Policy Manager 5.2. Refer the section titled "Policy Manager Details for Policy Manager 5.2" if that is the case.

- *security-url*
  URL of the Policy Manager UDDI V3 Security service. This property is optional and should only be specified if the service is not at the standard path relative to the registry-host URL. Default is "<registry-host>/uddi/security"

- *inquiry-url*
  URL of the Policy Manager UDDI V3 Inquiry service. This property is optional and should only be specified if the service is not at the standard path relative to the registry-host URL. Default is "<registry-host>/uddi/inquiry"

- *publish-url*
  URL of the Policy Manager UDDI V3 Publish service. This property is optional and should only be specified if the service is not at the standard path relative to the registry-host URL. Default is "<registry-host>:9901/uddi/publish"

- *technote-service-url*
  URL of the Policy Manager TechNote service. This property is optional and should only be specified if the service is not at the standard path relative to the registry-host URL. Default is "<registry-host>:9901/wsdltechnote"

- *policy-service-url*
  URL of the Policy Manager Policy Configuration Manager service. This property is optional and should only be specified if the service is not at the standard path relative to the registry-host URL. Default is "<registry-host>/policyconfiguration"

- *contract-service-url*
  URL of the Policy Manager Contracts Manager service. This property is optional and

[2] The user specified should have admin privileges in Policy Manager.

should only be specified if the service is not at the standard path relative to the registry-host URL. Default is “<registry-host>/contract”

- *compliance-service-url*
  URL of the Policy Manager Compliance service. This property is optional and should only be specified if the service is not at the standard path relative to the registry-host URL. Default is “<registry-host>/compliance”

- *workflow-service-url*
  URL of the Policy Manager Workflow service. This property is optional and should only be specified if the service is not at the standard path relative to the registry-host URL. Default is “<registry-host>/workflowservice”

- *repository-integration-service-url*
  URL of the Repository Integration Manager service. This property is optional and should only be specified if the service is not at the standard path relative to the registry-host URL. Default is “<registry-host>/repository”

- *rest-url*
  URL of the Policy Manager REST servlet. This property is optional and should only be specified if the servlet is not at the standard path relative to the registry-host URL. It is used to construct URLs to certain object in Policy Manager. Default is “<registry-host>/rest/”

- *keystore-service-url*
  URL of the Policy Manager KeyStore service. This property is optional and should only be specified if the service is not at the standard path relative to the registry-host URL. It is used to retrieve certificates needed to communicate with other Policy Manager services. Default is “<registry-host>/KeyStoreService”

Example Configuration

```
<federated-systems>
  <federated-system name="SMTest" class="PolicyManager">
    <properties>
       <property name="registry-host" value="http://example.com:9900"/>
       <property name="user" value="administrator"/>
       <property name="password" value="password" encrypt="true"/>
       <property name="version" value="6.0"/>
     </properties>
  </federated-system>
</federated-systems>
```

Synchronizing Groups and Organizations

Publishing of services into Policy Manager requires that Repository/Lifecycle Manager Groups and Policy Manager organizations be synchronized. This is accomplished in two parts: a

SynchronizePolicyManagerOrganizations command to synchronize existing Repository/Lifecycle Manager Groups[3] and a PolicyManagerOrganizationPublisher listener for ongoing synchronization of groups. These are described below:

SynchronizePolicyManagerOrganizations

- **Purpose:**
  Command for creating Policy Manager Organizations to reflect Repository/Lifecycle Manager groups.

- **Parameters:**
  Parameter 1: name of the Policy Manager federated-system to synchronize with.

Notes

- The library name will be used to name the Organization associated with Enterprise Group.
- The following Services and related Operational Policies need to be setup in Policy Manager for this command to execute successfully
    - Compliance Service - PolicyManagerDefaultUsernameTokenSecurityPolicy
    - Contracts Manager - PolicyManagerDefaultUsernameOrX509TokenSecurityPolicy
    - PolicyConfigurationmanagerservice - PolicyManagerDefaultUsernameTokenSecurityPolicy
    - Repository Integration Manager Service - PolicyManagerDefaultUsernameTokenSecurityPolicy
    - WSDL Tech Note - PolicyManagerDefaultUsernameTokenSecurityPolicy
    - Workflow Service - PolicyManagerDefaultHttpSecurityPolicy
    - UDDI Inquiry – No Operational Policy needs to be defined.
    - UDDI Publish – No Operational Policy needs to be defined.
    - UDDI Security – No Operational Policy needs to be defined.

[3] The Enterprise Group needs to be synchronized even in the case of a new library.

PolicyManagerOrganizationPublisher

- **Behavior:**
  Publishes and updates Repository/Lifecycle Manager Groups as Organizations in a specified Policy Manager installation.

- **Usage Context:**
  Generally configured to be triggered at Group create/update/delete time by the following events:
  ORGGROUP_CREATED
  PROJECT_CREATED
  ORGGROUP_DELETED
  PROJECT_DELETED
  ORGGROUP_UPDATED
  PROJECT_UPDATED

- **Class:** com.logiclibrary.listeners.PolicyManagerOrganizationPublisher

- **Properties:**

  - *federated-system-name*
    This is the name of the Policy Manager federated-system to publish to (Required)

- **Prerequisites:**
  The Organization in Policy Manager must not have child Organizations or services. In this case, an error[4] will occur resulting in a -1 return code from the listener.

- **Return Codes:**

  - *0 – success*

Notes

The following listener, filter and action needs to be present in the Library Process Configuration file (lpc) for the ongoing synchronization of Groups between Lifecycle Manager and Policy Manager.

In the Global Listeners section define the following listener:

```
<listener name="SM Organization Publisher"
class="PolicyManagerOrganizationPublisher">
      <properties>
        <property name="federated-system-name" value="SM60Dev2" />
```

[4] Currently it is not possible for RepositoryManager to distinguish this error from other more general system errors, so is handled only as a -1 return code.

```
      </properties>
</listener>
```

In the Global Filters section define the following filter:

```
<filter name="Group Events">
      <event>ORGGROUP_CREATED</event>
      <event>PROJECT_CREATED</event>
      <event>ORGGROUP_DELETED</event>
      <event>PROJECT_DELETED</event>
      <event>ORGGROUP_UPDATED</event>
      <event>PROJECT_ACTIVATED</event>
</filter>
```

In the Global Actions section define the following action:

```
<action name="Synchonize Groups">
    <trigger-event>
        <event-filter>Group Events</event-filter>
    </trigger-event>
    <listener>SM Organization Publisher</listener>
</action>
```

Publishing Services

The integration supports the publishing of Repository/Lifecycle Manager Assets representing web services[5] into Policy Manager. Such assets may contain a WSDL document as an artifact. This WSDL should define a single web service. Publishing and updating of service assets into Policy Manager is the responsibility of the PolicyManagerPublisher listener.

When a service is published into Policy Manager, a businessService object will be created reflecting the name and version of the Repository/Lifecycle Manager asset. Classifiers with defined mappings in the GDT will be populated as categorizations on the Policy Manager businessService as described in Appendix A below. If present, the asset's WSDL document will be associated with the Policy Manager businessService. Policy Manager will then populate any referenced schemas.

Service Updates

The PolicyManagerPublisher listener may be configured to handle service asset updates into Policy Manager. On a service update the following actions may be taken:

- The businessService name, version, and description updated
- The businessService categorizations updated

[5] Specifically, assets representing complete web services (single-asset representation) and those assets representing a service implementation (multi-asset representation). Service interface assets and associated schema assets are currently not explicitly published to Policy Manager.

- The WSDL document updated

PolicyManagerPublisher Listener Details

- **Behavior:**
  Publishes and updates web service Assets into a specified Policy Manager installation. This listener will construct a WSDL definition based on artifacts from the asset. If there isn't enough information or a supporting schema cannot be found, errors may occur during publish. The asset can contain either a WSDL or ZIP containing a WSDL, possibly with supporting schemas. If the asset contains a WSDL, support schemas will be looked up based on namespace classifier information and will fallback to schemaLocation information if needed. See the LPC guide Appendix X on WSDL / Schema resolution for more details on how this works. If the asset contains a ZIP file, supporting schemas can be packaged within the ZIP file. When publishing an artifact of type *wsdl-category* or *packed-service-artifact-category* should exist containing the (zipped) WSDL.

  Classifier Mapping
  Classifiers can be mapped to the Policy Manager service as UDDI tmodels. This is achieved by assigning an external mapping in the Global Definition Template inside the define-classifier or define-compound-classifier element with the "Policy Manager" mapping id. The tmodel must be a defined category in Policy Manager. See the example GDT section below.

  Artifact Mapping
  Artifacts can also be mapped into Policy Manager as attachments. Artifacts can be by-value, by-reference, or by-description, but only one type for an artifact should be used as the containment cannot be changed on Policy Manager. It is recommended to specify the containment type in the artifact definition to scope the containment to only one particular type. See the example GDT section below.

- **Usage Context:**
  Generally configured to be triggered at Asset publish/republish time by the *ASSET_AUTO_PUBLISH*, *ASSET_MANUAL_PUBLISH*, *ASSET_AUTO_REPUBLISH*, and *ASSET_MANUAL_REPUBLISH* Events. Should be restricted with a Filter allowing only Assets that represent web services.

- **Class:** com.logiclibrary.listeners.PolicyManagerPublisher

- **Properties:**

  - The WSDL / schema resolution properties are supported for this importer. See Appendix X of the Library Process Configuration guide for the full set.

- *federated-system-name*
  This is the name of the Policy Manager federated-system to publish to (Required)

- *publish-wsdl*
  Determines whether or not the WSDL is published along with the service. This may be used to support workflows, where the service is modified in policy manager and you do not wish to overwrite those changes. This property is optional. If not specified, this property defaults to "true".

- *email-exception*
  A boolean that determines whether or not to email the submitter of Policy Manager publish errors. Default: false.

- *email-administrators*
  A boolean that determines whether or not to email the library administrators of Policy Manager publish errors. The email-exception property must also be enabled. Default: false.

- *log-count*
  The number of logs to attach to the email when an exception occurs. The most recent logs will be attached. The email-exception property must also be enabled. Default: 2.

- *message-id*
  A message id that will be used for any email resulting from an exception occurring during publish. The email-exception property must also be specified. See the "Commands" section in the RepositoryManager System Administration guide for more information about retrieving mail template information. The default is a built in message. The substitution parameters you can use to customize the message are:

  0: Asset Name
  1: Asset Version
  2: Asset ID
  3: Submitter account
  4: Exception

- **Prerequisites:**
  Groups in the RepositoryManager library must be synchronized with the Policy Manager installation. The Qname of the the service being published (this WSDL namespace qualified service name) must be unique within the Policy Manager installation.

- **Return Codes:**

  - *0 – success*

  - *1 – Duplicate service error*

- *2 – Unsynchronized owning Group*

Example GDT configuration

The following XML snippet depicts three elements in the GDT (define-classifier, define-compound-classifier, and define-artifact). These mappings would cause classifier or artifact information to be published to the Policy Manager service as service categories or attachments, respectively.

```
<define-classifier name="aggregated-policy-instance" type="string" >
      <external-mapping key="Policy Manager" value="uddi:acme.com:techpolicy"
      />
</define-classifier>
<define-compound-classifier name="business-domain" open="false" max-
occurs="1">
      <fields>
            <field name="general" />
            <field name="specific" />
      </fields>
      <external-mapping key="Policy Manager"
value="uddi:soa.com:rm:businessdomain" />
      <add-value value="Shipping|Request">
            <external-mapping key="Policy Manager" value="Shipping|Request"
name="Shipping Request Business Domain" />
      </add-value>
      <add-value value="Shipping|Quote">
            <external-mapping key="Policy Manager" value="Shipping|Quote"
name="Shipping Quote Business Domain" />
      </add-value>
</define-compound-classifier>

<define-artifact category="doc" display-name="Documentation" containment="by-
value">
      <external-mapping key="Policy Manager" value="Documentation" />
</define-artifact>
```

Contracts

The PolicyManagerContractPublisher listener may be used to create Policy Manager consumer contracts to correspond with Repository/Lifecycle Manager asset acquisitions.

PolicyManagerContractPublisher Listener Details

- **Behavior:**
  Publishes Policy Manager consumer contracts between consuming and producing organizations for a service. Allows assignment of a service-level policy through the use of a service level agreement on the contract. Other aspects of the contract that can be set include contract

anonymity, user identities, and consumed operations. Multiple contracts can be created during an acquisition. See the nested property details bullet below.

- **Usage Context:**
  Generally configured to be triggered at Asset acquisition time by the *ASSET_ACQUISITION_APPROVED* event. Should be restricted with a Filter allowing only Assets that represent web services.

- **Class:** com.logiclibrary.listeners.PolicyManagerContractPublisher

- **Properties:**

  - *federated-system-name*
    This is the name of the Policy Manager federated-system to publish to (Required)

  - *description*
    The description to be used for the Policy Manager contract. This property is optional. If not specified, the contract description is defaulted to "Contract created via Repository/Lifecycle Manager acquisition".

  - *start-time-request-property*
    The name of the property on the acquisition AssetRequest to use as the start time for the contract. This property is optional. If not specified, the "contract-effective-date" request property is used.

  - *end-time-request-property*
    The name of the property on the acquisition AssetRequest to use as the expiration time for the contract. This property is optional. If not specified, the "contract-expiration-date" request property is used.

  - *service-level-policy-property*
    The name of the property on the asset acquisition request used to hold an optional key to a service-level policy to associate with the contract. This property is optional. If not specified, the "service-level-policy" request property is used. The request property is normally tied to the PolicyManagerSLPs value source.

  - *constraint-policy-property*
    The name of the property on the asset acquisition request used to hold a list of additional policies to associate with the contract. This property is optional. If not specified, the "constraint-policy" request property is used. The request property is normally tied to the PolicyManagerSLPs value source.

  - *contract-reference-property*
    The name of the property on the asset acquisition request to use for storing a URL to

the Policy Manager contract detail page. This property is optional. If not specified, the "contract-reference" request property is used.

- *consumed-service-key-property*
  The name of the property on the asset acquisition request to use for the consumed service of the contract. Normally this property will be connected to a VirtualServiceValueSource, so the user can select which virtual service they are acquiring. This property is optional. If not specified, the "consumed-service" request property is used.

- *user-identity-property*
  The name of the property on the asset acquisition request to use on the contract to reduce consumption scope of the contract. Normally this property will be connected to a PolicyManagerIdentityValueSource, so the user can select the consuming user. This property is optional. If not specified, the "user-identity" request property is used.

- *sla-name-pattern*
  Determines the name to use for the SLA when attaching an optional service-level policy. A substitution approach is used with "{SLP_NAME}" representing the name of the service-level policy. This property is optional. If not specified, the default value is "{SLP_NAME}" which simply uses the service-level policy name as the SLA name.

- *sla-description-pattern*
  Determines the description to use for the SLA when attaching an optional service-level policy. A substitution approach is used with "{SLP_NAME}" representing the name of the service-level policy. This property is optional. If not specified, the default value is "Generated Service Level Agreement for {SLP_NAME} Service Level Policy".

- *consumed-operation-property*
  Specifies the name of a property that represents consumed operations, restricting the scope of the contract these operations. This property will usually be sourced from a ServiceOperationsValueSource that displays the service's operations in a user friendly format. Default: consumed-operation

- *anonymous-contract-property*
  Specifies the name of a Boolean property that determines whether the contract created is anonymous or not. If the contract is anonymous, then no consuming organization or service is applicable to the contract. Default: anonymous-contract

- *update-contracts*
  A Boolean (true/false) indicating whether to update existing contracts when acquisition properties are updated. Contracts will either be created, updated, or revoked depending if the existing contracts are still applicable. Default: false

- *activate-contract-workflow-action*
  Specifies the name of a Policy Manager workflow action to run when a new contract is created or when a new version is created for an existing contract. This property only applies if *update-contracts* is set to true. If this property is not specified then no action will be taken on the new contract(s) and will be left in a draft state.

- *revoke-contract-workflow-action*
  Specifies the name of a Policy Manager workflow action to run when an existing contract is no longer applicable to the acquisition. This property only applies if *update-contracts* is set to true. If this property is not specified then contracts will not be revoked – this is probably not what is intended so it should be a valid workflow action that can be called to revoke the contract.

- **Nested property details:**
  The Policy Manager contract can be created in several ways depending on how you configure the properties. If the service-level-policy-property contains nested Policy Manager specific properties (eg. consumed-operation), then one contract will be created for each SLP specified on the request, and the contract will be attached to the SLP property instead of the asset request. If the constraint-policy-property is a nested property underneath the service-level-policy-property all the nested constraint policies will be added to the contract.

- **Prerequisites:**
  Groups in the Repository/Lifecycle Manager library must be synchronized with the Policy Manager installation. The asset being acquired must have a corresponding businessService in Policy Manager.

- **Return Codes:**
  - *0 – success*
  - *1 – No corresponding businessService*
  - *2 – Unsynchronized Group*

Lifecycle

State transitions of Policy Manager services and contacts may be triggered from Repository/Lifecycle Manager through use of the PolicyManagerTransition listener.

PolicyManagerTransition Listener Details

- **Behavior:**
  Performs a lifecycle transition for a specified service or contract within Policy Manager.

- **Usage Context:**
  Assets: Generally configured to be triggered at Asset republish time by the *ASSET_AUTO_REPUBLISH* and *ASSET_MANUAL_REPUBLISH* Events. Commonly a transition will

be associated with an AssetFilter that triggers the transition when the asset's classifiers match specified criteria. Should be restricted with a Filter allowing only Assets that represent web services.
Contracts: Configured to be triggered on revocation of an asset registration in Repository/Lifecycle Manager using the ASSET_DEREGISTERED event.

- **Class:** com.logiclibrary.listeners.PolicyManagerTransition
- **Properties:**
    - *federated-system-name*
      This is the name of the Policy Manager federated-system to publish to (Required)
    - *object-type*
      Determines whether the target of the transition is a service or a contract. Valid values are "service" and "contract" (Required).
    - *action*
      The name of the action (transition) to be applied. For example "Mark as Obsolete" (Required).
    - *message*
      The message stored with the transition in the service or contract's lifecycle history. This property is optional. If not specified, "Action requested by Repository/Lifecycle Manager" is used as the message.
- **Prerequisites:**
  A corresponding businessService or contract must exist in Policy Manager with a defined workflow. The specified action must be valid for the current state of the businessService or contract. In the case of an asset registration (contract) the acquisition Asset Request for the registration must exist and must have been the target of the PolicyManagerContractPublisher listener.
- **Return Codes:**
    - *0 – success*
    - *1 – No corresponding Policy Manager businessService*
    - *2 – Workflow not defined for businessService or contract*
    - *3 – Invalid action*
    - *4 – Contract could not be retrieved for asset registration*

Service Metrics

Synchronization of Policy Manager service metrics back into Repository/Lifecycle Manager as asset properties is possible with the PolicyManagerRuntimeStatistics listener.

PolicyManagerRuntimeStatistics Listener Details

- **Behavior:**
  Copies tModel information from a Policy Manager service back to its corresponding Repository/Lifecycle Manager asset. The tModel are mapped to asset properties using a comma-separated value file format. When the listener runs it identifies all Policy Manager service mapped assets in Repository/Lifecycle Manager and updates the asset's properties with values corresponding to the service's tModel categories. A failure to update an asset for any reason does not terminate the listener and will continue to process other assets. Only one property per tModel key is supported, so if you have multiple tmodels with the same key in Policy Manager, only one property will be populated for the Asset.

- **Usage Context:**
  Assets: Generally configured to be triggered using a timer-based approach, i.e. every day.

- **Class:** com.logiclibrary.listeners.PolicyManagerRuntimeStatistics

- **Properties:**

  - *federated-system-name*
    This is the name of the Policy Manager federated-system that will be synchronized (Required)

  - *tmodel-mappings-document-id*
    The name of the a comma-separated value file (CSV) containing one row per tModel/property mapping. See examples below for format information (Required).

- **Prerequisites:**
  The federated system should be setup and at least one service should be published to Policy Manager.

- **Return Codes:**

  - *0 – success*

- **Example CSV file:**
  The following file demonstrates how to map two service tModels to asset properties.

```
uddi:systinet.com:management:metric:errors,uddi-metric-errors
uddi:systinet.com:management:metric:hits,uddi-metric-hits
```

Validation

PolicyManagerServiceReferenceClassifierValidator Details

- **Behavior:**
  Used to identify the asset being edited with an existing Policy Manager service. This allows the user to set a classifier corresponding with a service in Policy Manager which will then be associated with the asset in Lifecycle manager. The service and wsdl artifact source links will then work for the service selected.
- **Class:** com.logiclibrary.registry.polm.PolicyManagerServiceReferenceClassifierValidator
- **Properties:**
  - *federated-system-name*
    This is the name of the Policy Manager federated-system to associate the asset with. (Required)
  - *classifier-name*
    The name of the classifier containing the service reference – this should be sourced from the PolicyManagerServiceValueSource

PolicyManagerServiceReferenceRelationshipValidator Details

- **Behavior:**
  Used to populate endpoint information for an asset based on a relationship property to a Policy Manager service. This validator is mainly used to support the use case where a virtual service is selected during a property selection process and the virtual service endpoint is copied onto the asset. This also allows the asset to be associated with the Policy Manager service so Policy Manager artifact sources work to link to the Policy Manager service and WSDL.
- **Class:** com.logiclibrary.registry.polm.PolicyManagerServiceReferenceRelationshipValidator
- **Properties:**
  - *federated-system-name*
    This is the name of the Policy Manager federated-system to query.
    (Required)
  - *endpoint-classifier-name*
    The classifier name that the endpoint information is populated into.
    (Default: endpoint)
  - *relationship-name*
    The name of the relationship containing the properties to query. (Default: implements)

- *relationship-property*
  The name of the property containing the virtual service.

PolicyManagerValidator Details

- **Behavior:**
  Used to test artifacts using Policy Manager Compliance Validation. This validator can only handle standalone schemas and WSDLs that only contain references to other resources with absolute URIs. See the PolicyManagerWSDLValidator for enhanced WSDL processing support. The validator tests each target artifact against the compliance rules setup for the Policy Manager organization corresponding to the asset's owning group.

- **Usage Context:**
  Used during asset publish.

- **Class:** com.logiclibrary.listeners.PolicyManagerValidator

- **Return Codes:**
  1 – Success
  2 – Failure with warnings
  3 – Failure with errors
  99 – Artifact not found

- **Properties:**

  - *federated-system-name*
    This is the name of the Policy Manager federated-system to publish to.
    (Required)

  - *target-artifact-category*
    This is the artifact category name containing the artifacts to validate.
    (Required)

  - *result-artifact-category*
    This is the name of an artifact category that will contain the results of the Policy Manager compliance validation.

  - *exclude-successful-results*
    If validation is successful and this property is set to "true", then the Policy Manager compliance report is not attached to the asset.

  - *validation-succeeded*
    The name of a classifier that will be set to either "true" or "false" depending on whether the validation was successful or not.

- *locking-user*
  The name of a user that will be used to make updates to the asset. If omitted the default is the "Repository Manager" user

- *replace-results*
  If replace results is set to true, any existing Policy Manager validation results are removed prior to attaching the results of the most recent execution. The default is "false".

- *policy-target*
  This is the name of the Policy Manager policy target to use when validating. Potential values could be "schema" or "wsdl". If this value is omitted Repository/Lifecycle Manager will attempt to determine the type based on the artifacts being submitted for validation.

PolicyManagerWSDLValidator Details

- **Behavior:**
  Used to test artifacts using Policy Manager Compliance Validation. This validator can handle WSDLs packed in a ZIP file, or standalone. It can also handle WSDLs with imports based on namespace (assuming the imported WSDL/Schemas contain appropriate target namespace classifiers. The validator tests each target artifact against the compliance rules setup for the Policy Manager organization corresponding to the asset's owning group.

- **Usage Context:**
  Used during asset publish.

- **Class:** com.logiclibrary.listeners.PolicyManagerWSDLValidator

- **Return Codes:**
  1 – Success
  2 – Failure with warnings
  3 – Failure with errors
  99 – Artifact not found

- **Properties:**

  - *federated-system-name*
    This is the name of the Policy Manager federated-system to publish to.
    (Required)

  - *target-artifact-category*
    This is the artifact category name containing the artifacts to validate.
    (Required)

- *result-artifact-category*
  This is the name of an artifact category that will contain the results of the Policy Manager compliance validation.

- *exclude-successful-results*
  If validation is successful and this property is set to “true”, then the Policy Manager compliance report is not attached to the asset.

- *validation-succeeded*
  The name of a classifier that will be set to either “true” or “false” depending on whether the validation was successful or not.

- *locking-user*
  The name of a user that will be used to make updates to the asset. If omitted the default is the “Repository Manager” user

- *replace-results*
  If replace results is set to true, any existing Policy Manager validation results are removed prior to attaching the results of the most recent execution. The default is “false”.

- *policy-target*
  This is the name of the Policy Manager policy target to use when validating. If this value is omitted Repository/Lifecycle Manager will use “wsdl”.

- *service-artifact-category*
  The name of the artifact category that the publisher will use for access to the WSDL document. This property is optional. If not specified, the WSDL artifact category is defaulted to “message-definition”.

- *schema-artifact-category*
  The name of the artifact category to use when searching for supporting schemas assets. See the namespace-classifier property as well. This property is optional. If not specified, the “schema-definition” category is used.

- *namespace-classifier*
  The name of the classifier that contains the target namespace of supporting schemas/services. When searching for supporting schemas by namespace, assets matching this classifier will be used. If multiple assets contain the same namespace, an error occurs. If not specified, the “target-namespace” classifier is used.

- *packed-service-artifact-category*
  The name of the artifact category containing a zipped WSDL and its associated schemas. If not specified, the “packed-service” artifact category is used.

Artifact Source

A PolicyManagerArtifactSource is defined to allow assets to reference Policy Manager service detail pages and WSDL documents.

PolicyManagerArtifactSource Details

- **Behavior:**
  Used for displaying either service detail pages or retrieving WSDL documents from Policy Manager. In the WSDL case, the artifact source will attempt to first locate the actual consumed Policy Manager service from an existing Consumption Context between the currently active Asset (the "consuming" Asset) and the Asset being viewed (the "consumed" asset). If a Consumption Context cannot be found, or if there is no consumed service property set on the Context, the WSDL of the associated Policy Manager physical service will be retrieved.

- **Usage Context:**
  Used on assets representing web services.

- **Class:** com.logiclibrary.artifact.sources.PolicyManagerArtifactSource

- **Properties:**

  - *federated-system-name*
    This is the name of the Policy Manager federated-system to publish to
    (Required)

  - *consumed-service-key-property*
    This the property name to use in retrieving the consumed Policy Manager service key from the consumption context. The default property name is "consumed-service"
    (Optional)

  - *consuming-relationship-property*
    This is the name of the consuming relationship that may exist from the active asset to the service asset being viewed. The default value for this property is "service-used"
    (Optional).

Value Source

Repository/Lifecycle Manager allows valid values for Asset Requests to be source from an external system such as Policy Manager. Specifically, a PolicyManagerSLPs value-source is provided to provide a list of defined service-level policies in a Policy Manager installation.

PolicyManagerDynamicIdentityValueSource

- **Applicability**

  Classifiers

- **Resolution**

  Dynamic

- **Behavior:**
  When creating non-service assets, there are scenarios that can happen on behalf of an application "user". These users are sourced from LDAP. What this value source does is search LDAP for candidate users. By default the LDAP server that is queried is the LDAP server Lifecycle Manager is using. However, this can be configured to use any LDAP server. However a Policy Manager identity source (domain) and this value source must both be configured for the same LDAP server. This class is meant to work in conjunction with the PolicyManagerIdentityValidator, which populates the identity on the asset's owning organization in Policy Manager, and the PolicyManagerIdentityValueSource, which consumes the identity on the acquisition request when acquiring an asset.

- **Class:** com.logiclibrary.registry.polm.PolicyManagerDynamicIdentityValueSource

- **Properties:**

  - *federated-system-name*
    This property is used to specify the Policy Manager federated system that is contacted to enumerate the user identities. This should correspond with the same system that was used to create the physical service in Policy Manager.

  - *domain*
    This is the Policy Manager domain corresponding to the LDAP server that is being used. In Policy Manager, the domain precedes the username. For example in "corp\user", corp is the domain.

  - *ldap-url*
    The LDAP server to use when enumerating the list of users. If this is not specified the Lifecycle Manager LDAP server will be used. The format of this string corresponds with the format of Lifecycle Manager's installation LDAP URL format (see the administrator guide for more details). The format is
    ldap://host:port/basedn?useridattribute?searchscope?filter
    E.g.: ldap://example.com:389/ou=people,dc=example,dc=com?uid

  - *ldap-bind-dn*
    This is the user to bind to the LDAP server to perform searches. This is optional and only

applicable if *ldap-url* is specified.
E.g. uid=serviceuser,ou=people,dc=example,dc=com

- *ldap-bind-password*
  This is the password for the corresponding user above. This is optional and only applicable if *ldap-url* is specified.

- *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

- ### *Example Configuration*

```
<value-source name="PolicyManagerDynamicIdentities"
class="com.logiclibrary.registry.polm.PolicyManagerDynamicIdentityValueSource
">
  <properties>
    <property name="ldap-url"
        value="ldap://example.soa.local/ou=People,dc=example,dc=com" />
    <property name="ldap-bind-dn"
        value="cn=manager,ou=people,dc=example,dc=com" />
    <property name="ldap-bind-password" value="password" />
    <property name="federated-system-name" value="SMTest" />
    <property name="search-field" value="Last Name" />
    <property name="domain" value="corp" />
  </properties>
</value-source>
```

PolicyManagerIdentityValueSource

- **Applicability**

  Properties

- **Resolution**

  Dynamic

- **Behavior:**
  When acquiring assets in an Asset Based Acquisition scenario, this value source can limit the consumption scope of the contract with a user identity. This only applies to acquiring assets that do not exist in Policy Manager (application assets for example). The list of identities that are listed are those corresponding to the Policy Manager group of the acquiring asset’s owning group. For project based acquisitions, the only selection available will be “Not Applicable”. For acquiring assets (services) that already exist in Policy Manager, the only selection available will be “Use Service”, in either of these two cases, it will not affect the consumption scope of the contract.

- **Class:** com.logiclibrary.registry.PolicyManagerIdentityValueSource
- **Properties:**
    - *federated-system-name*
      This property is used to specify the Policy Manager federated system that is contacted to enumerate the user identities. This should correspond with the same system that was used to create the physical service in Policy Manager.
    - *identity-classifier-name*
      This property is used to specify a classifier that identifies the acquiring asset's consumer identity in Policy Manager. This value is usually sourced from the PolicyManagerDynamicIdentityValueSource. If an organization identity is found with this value, then the user will only be presented with this value, otherwise the full list of identities for the owning group is presented.
      Default: policy-manager-identity

Example Configuration

By default, when a service in Repository/Lifecycle Manager is acquired, the underlying a contract will be created in Policy Manager. To add a user identity to the contract, specify a "user-identity" property that is populated from this value-source. This would resemble:

```
<property-definition name="user-identity" value-
source="PolicyManagerIdentities" type="STRING" display-name="Policy Manager
User Identity" help-text="Policy Manager Identity for the consumption scope
of the contract" target="ASSET_REQUEST"/>
```

The property needs to be added to the corresponding acquisition property constraint sets, along with configuring a value-source called "PolicyManagerIdentities" (if using the example above).

PolicyManagerServicesValueSource

- **Applicability**

  Classifiers / Properties

- **Resolution**

  Dynamic

- **Behavior:**
  Used for displaying a list of Policy Manager services in organizations that are mapped to Lifecycle Manager groups.

- **Usage Context:**
  Used for scenarios with hooking up an existing virtual service to an asset. See the validator PolicyManagerServiceReferenceClassifierValidator.

- **Class:** com.logiclibrary.registry.polm.PolicyManagerServicesValueSource
- **Properties:**
    - *federated-system-name*
      This is the name of the Policy Manager federated-system to query
      (Required)
    - *category-filter**
      Any property that starts with category-filter will be considered a category bag filter and only those services containing the category will be returned. The value of this property should either be "*tmodelkey"* or "*tmodelkey=value*". In the first case, it will simply check for the existence of the tmodel category, in the second it will require that the tmodel category exists with a particular value. For instance, this filter value will only display virtual services "uddi:digev.com:categorization:services=Virtual Service".

PolicyManagerSLPs Details

- **Applicability**

  Properties

- **Resolution**

  Dynamic

- **Behavior:**
  Used for displaying a list of valid property values based on the defined service-level policies in a Policy Manager installation. Presents service-level policy names, descriptions, and detail page URLs.

- **Usage Context:**
  Used on asset acquisition requests for web service assets.

- **Class:** com.logiclibrary.registry.PolicyManagerSLPs

- **Properties:**
    - *federated-system-name*
      This is the name of the Policy Manager federated-system to publish to
      (Required)
    - *policy-types*
      A comma-delimited list of Policy Manager QoS policy types to display. You can also exclude policies from being included, using a minus sign in front of the policy type, e.g. "-policy.qos.sla". If you specify an exclude then all policies but the excluded ones will be

displayed. Default: Service Level Policies (qos.sla.policy). Valid values and their Policy Manager types are:
policy.qos.bandwidthquota – Bandwidth Quota Policy
policy.qos.concurrencyquota – Concurrency Quota Policy
policy.qos.scripted – Script Policy
policy.qos.slaenforcement – Service Level Enforcement Policy
policy.qos.sla – Service Level Policy
policy.qos.throughputquota – Throughput Quota Policy
policy.qos.timeout – Timeout Policy

ServiceOperationValueSource

- **Applicability**

  Properties

- **Resolution**

  Dynamic

- **Behavior:**
  Displays operations for a service in Policy Manager. The operations can either be listed for the consumed service or the consuming service in Policy Manager. The Policy Manager Contract publisher can use this value source to define which operations are then valid for a contract in either the consuming or consumed side of the contract. The policies enumerated will be found by locating the Policy Manager organization belonging to the service and storing all policies at that level and each parent org up the hierarchy.

- **Properties:**

  - *federated-system-name*
    This property is used to specify the *Policy Manager federated system* that is contacted to enumerate the operations for the service. This should correspond with the same system that was used to create the service in Policy Manager.

  - *consuming-asset*

    This Boolean property determines whether the operations are listed for the service being acquired if the property is set to “false”, or for the acquiring service if the property is set to “true”.

Example Configuration

```
<value-source name="PolicyManagerConsumingOperations"
class="com.logiclibrary.registry.polm.ServiceOperationValueSource">
      <properties>
        <property name="federated-system-name" value="SMTest" />
```

```
        <property name="consuming-asset" value="true" />
      </properties>
</value-source>
```

VirtualServiceValueSource

- **Applicability**

  Properties

- **Resolution**

  Dynamic

- **Behavior:**
  Displays potential Policy Manager services based on the asset being acquired. A service being acquired may be associated with a service in Policy Manager. If this service has associated proxies, then this Value Source will return the available virtual service proxies for this service, or other virtual service proxies of the proxy. If there are no proxies, then the physical service will be the only available value. This behavior may be modified using the *display* property below.

- **Properties:**

  - *federated-system-name*
    This property is used to specify the Policy Manager federated system that is contacted to enumerate the virtual services for the physical service. This should correspond with the same system that was used to create the physical service in Policy Manager.

  - *display*
    This property is used to specify what services are displayed for the value source.

    - *all – Lists the physical service along with all virtual services*
    - *default – Lists only virtual services if they exist. If there are no virtual services, then it will list the physical service.*
    - *virtual – Lists only the virtual services corresponding to the physical service.*

  - *organization-filter*
    This property is constrains the list of services that are displayed for the value source. If it is unset any proxied virtual service will be displayed. If it is set to “physical-service”, only virtual services that have the same parent organization as the physical service will be displayed.

Example Configuration

By default, when a service in Repository/Lifecycle Manager is acquired, the underlying a contract will be created in Policy Manager associated with the corresponding physical service. To modify the acquisition process to use a virtual service proxy, a property definition for "consumed-service" should be changed/added to use the Virtual Service Value Source. This would resemble:

```
 <property-definition name="consumed-service" value-
source="PolicyManagerVirtualServices" type="STRING" display-name="Policy
Manager Service" help-text="Physical or Virtual Service acquisition"
target="ASSET_REQUEST"/>
```

The property needs to be added to the corresponding acquisition property constraint sets.

Importers

Repository/Lifecycle Manager allows services to be imported from various external systems. The importers that follow allow services to be imported from Policy Manager into Lifecycle Manager, modified, and updated back into Policy Manager.

PolicyManagerImporter

- **Behavior:**
  The PolicyManagerImporter can be configured to import services from an already configured Policy Manager federated system. The asset in Lifecycle Manager will contain the corresponding WSDL (if one exists in Policy Manager) as a by-value artifact. Only those services that belong to mapped organizations between the two systems will be displayed for import, and the Lifecycle Manager asset's owning group will be corresponding organization from the Policy Manager service. If a Lifecycle Manager asset already exists for a Policy Manager service, the importer will flag it as an error and not be imported. The imported WSDL will either be a standalone WSDL or a ZIP of all resources used to construct the complete WSDL. The name of the asset will be the name of the service, and the corresponding artifact source links will be active as appropriate (if there is no Policy Manager WSDL, the WSDL link will not be active).
  Caveats: The search box should not contain wildcards. The search will be performed using a substring match against the entered text.
- **Class:** com.logiclibrary.registry.polm.PolicyManagerImporter
- **Properties:**
  - *federated-system-name*
    This property is used to specify the Policy Manager federated system that is contacted to enumerate the services. This should correspond with an already existing federated system that Lifecycle Manager was synchronized with.

- *service-asset-template*
  The template name of the service that will be used for the service assets that are created. Default: Service

- *service-asset-type*
  The asset-type classifier of the service that will be used for the service assets that are created. Default: Service

- *service-asset-version*
  This is the version of the service asset that will be used. Default: 1.0

- *wsdl-category*
  For Policy Manager services composed of only one WSDL resource this is the artifact category that the WSDL will be created under. Default: message-definition

- *packed-service-artifact-category*
  For Policy Manager services composed or more than one resource (e.g. a WSDL importing an XSD), this is the artifact category that the zipped service will be created under. Default: packed-service

- *import-physical-services*
  A Boolean indicating whether or not physical services will be allowed for import. If this is set to false, then any attempt to import a physical service will result in an import error. Default: true.

- *import-virtual-services*
  A Boolean indicating whether or not virtual services will be allowed for import. If this is set to false, then any attempt to import a virtual service will result in an import error. Default: false.

- *category-filter**
  Any property that starts with category-filter will be considered a category bag filter and only those services containing the category will be returned. The value of this property should either be “tmodelkey” or “tmodelkey=value”. In the first case, it will simply check for the existence of the tmodel category, in the second it will require that the tmodel category exists with a particular value. For instance, this filter value will only display virtual services “uddi:digev.com:categorization:services=Virtual Service”.

- *description*
  This is the description that is displayed in the user interface when the importer is selected from the dropdown list.

- *create-note*
  The note used when creating the asset

- *submit-note*
  The default submission note attached to the asset submission if using the "Import and Submit" option in the importer.

- ### *Example Configuration*

```
<importer name="Import Services from Policy Manager"
class="com.logiclibrary.registry.polm.PolicyManagerImporter">
  <properties>
    <property name="federated-system-name" value="SMTest" />
    <property name="service-asset-template" value="Complete Service
Specification - Development Complete" />
    <property name="service-asset-type" value="Service" />
    <property name="service-asset-version" value="1" />
    <property name="wsdl-category" value="message-definition" />
    <property name="packed-service-artifact-category" value="packed-service"
/>
  </properties>
</importer>
```

Synchronous Validators

Repository/Lifecycle Manager allows synchronous validation with various Policy Manager entities. The validation occurs whenever an asset is submitted for publish or when it is explicitly invoked by a user.

PolicyManagerIdentityValidator

- **Behavior:**
  When a non-service asset (e.g. Application) is created, an application user can be associated with the application that will be used in Policy Manager to associate an application identity to any contracts that are created for an acquiring application. This identity is sourced using the PolicyManagerDynamicIdentityValueSource and placed as a classifier on the asset. This validator will attach the user to the application's owning organization in Policy Manager.

- **Class:** com.logiclibrary.registry.polm.PolicyManagerIdentityValidator

- **Properties:**

  - *federated-system-name*
    This property is used to specify the *Policy Manager federated system* that is contacted to create an organization identity in Policy Manager.

  - *identity-classifier-name*
    The name of the identity classifier that contains the user identity that will be added as an organizational identity in Policy Manager.
    Default: policy-manager-identity

PolicyManagerSyncValidator

- **Behavior:**
  Before a service is published to Policy Manager, the validator can warn users of potential naming conflicts within Policy Manager. The validator is similar to the ServiceNamespaceValidator. This validator will check an asset's WSDL against Policy Manager to see if Policy Manager already contains 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.

- **Class:** com.logiclibrary.validators.PolicyManagerSyncValidator

- **Properties:**

  - *federated-system-name*
    This property is used to specify the *Policy Manager federated system* that is contacted to verify potential namespace collisions.

  - *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 Repository/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 Repository/Lifecycle Manager. The default is “target-namespace”.

Installation of Policy Manager (EJB) EAR

There are certain components of Repository/Lifecycle Manager that communicate with Policy Manager using an EJB that is deployed as an EAR file. This Policy Manager EJB exposes certain APIs that are not available through the normal Policy Manager interaction. If any of these components are utilized (which should be identified in this guide, under those components), then the EAR needs to be created and installed.

Follow these instructions to generate and install the EAR.

1. Copy the *PM_HOME*/sm52/config/bootstrap.properties and dems.properties file of the Policy Manager system you are communicating with to the *RM_HOME*/conf directory
2. Run *RM_HOME*/bin/install build-policymanager-ear (The ear will be created in *RM_HOME*/deploy/policymanager.ear
3. Deploy the ear according to the application server you are using (paths may need to be adjusted accordingly)

JBoss

- Create a directory in *JBOSS_HOME*/server/default/deploy/policymanager.ear
- Unjar the ear file in that directory
- Edit *JBOSS_HOME*/server/*server_name*/conf/jboss-service.xml
    - Locate the NamingService mbean XML section and modify “CallByValue” from false to true
    - It should resemble: <attribute name="CallByValue">true</attribute>
- Edit *JBOSS_HOME*/server/*server_name*/deploy/ear-deployer.xml
    - Locate the EARDeployer mbean XML section and modify “CallByValue” from false to true
    - It should resemble: <attribute name="CallByValue">true</attribute>

WebSphere

- Using the WebSphere admin console, install the new ear file.

- Ensure the EAR file is deployed to the same application server as RM is installed on.

WebLogic

- Create a directory next to the RM deployed app directory, for example, /opt/bea/user_projects/domains/base_domain/applications/policymanager
- Unjar the ear in that directory
- Login to the WebLogic console and step through deploying a new ear file, pointing it at the directory you created above.

Appendix A: Classification

Category information on the businessService in Policy Manager is determined from classifiers on the Repository/Lifecycle Manager asset. This is facilitated through the use of "external-mapping" elements in the Repository/Lifecycle Manager Definition Template.

Mapping Classifiers to Categories

Defining an external-mapping sub-element of the define-classifier element binds the classifier to a particular category in Policy Manager. The "key" of the external-mapping element should be the mapping-id property that was assigned to the PolicyManagerPublisher listener (the default mapping-id for Policy Manager is "UDDI". The "value" of the external-mapping element should be the UDDI V3 key of the category as defined in Policy Manager (e.g. "UDDI:SOA.com:rm:certificationlevel"). In the case of a "keyword"[6], the term "KEYWORD" may be used as the external-mapping value attribute.

Note that only classifiers with external-mappings defined with the appropriate Policy Manager mapping-id will be translated to Policy Manager categories.

Mapping Classifier Values

If an external-mapping is defined only at the define-classifier level, values for the classifier will be used directly as values for the categorization in Policy Manager. This is useful for "open" categories, however it is often necessary to translate the values of the RepositoyManager classifier to defined values within the Policy Manager category. This is done through use of an external-mapping element added as a sub-element of the "add-value" element in a classifier definition. The mapping-id attribute of the external-mapping element should be set to the mapping-id of the PolicyManagerPublisher, while the value attribute should be set to the actual value to use for the category within Policy Manager.

[6] In UDDI terms, a categorization based on the "uddi-org:general_keywords" category.

Example Configuration

The following example maps the “certification-level” classifier to the “soa.com:rm:certificationLevel” category. The example uses “Policy Manager” as the mapping id. The values “none”, “Owning Group”, and “Enterprise” are explicitly mapped to category values in PolciyManager. The values “Line of Business ABC” and “Line of Business XYZ” have no mappings so will be used literally as category values.

```
<define-classifier name="certification-level" display-name="Certification
Level" type="string" open="false" max-occurs="1" value-ordering="GDT" help-
text="The reuse level for which this asset has been approved.">
<external-mapping key="Policy Manager"
value="uddi:soa.com:rm:certificationlevel" />
    <add-value value="None">
      <external-mapping key="Policy Manager" value="No Certification
Level|None" />
    </add-value>
    <add-value value="Owning Group">
      <external-mapping key="Policy Manager" value="Owning Group
Certification Level|Owning Group" />
    </add-value>
    <add-value value="Line of Business ABC" />
    <add-value value="Line of Business XYZ" />
    <add-value value="Enterprise">
      <external-mapping key="Policy Manager" value="Enterprise Certification
Level|Enterprise" />
    </add-value>
  </define-classifier>
```

Mapping Compound Classifiers

Standard (“dependent”) compound classifiers are mapped in a similar fashion as single-value classifiers, with each defined value optionally having an external-mapping defined.

However, the fields of an independent compound classifier are handled separately with respect to external-mappings. In the case of an independent compound classifier, each “field” element may have its own external-mapping sub-element. Similarly, each add-value element within a field may have an external-mapping.

Example

In the following example, the field “general” is mapped to the “soa.com:rm:businessdomain” Policy Manager category. The second field “specific” as no external-mapping and will have no effect on catergorization in Policy Manager.

```
<define-compound-classifier name="business-domain" display-name="Business
Domain" max-occurs="1" independent="true" value-ordering="GDT" help-text="The
business domain to which this asset applies">
  <fields>
    <field name="general" open="false" required="true" help-text="The general
domain of applicability">
      <external-mapping key="Policy Manager"
value="uddi:soa.com:rm:businessdomain" />
```

```
      <add-value value="Educational Services">
        <external-mapping key="Policy Manager" value="Educational
Services|EDU" />
      </add-value>
      <add-value value="Finance and Insurance">
        <external-mapping key="Policy Manager" value="Finance and
Insurance|FI" />
      </add-value>
      <add-value value="Manufacturing" />
      <add-value value="Professional, Scientific, and Technical Services" />
      <add-value value="Real Estate and Rental and Leasing" />
      <add-value value="Retail Trade" />
      <add-value value="Transportation and Warehousing" />
      <add-value value="Utilities" />
      <add-value value="Wholesale Trade" />
      <add-value value="Not Applicable" />
    </field>
    <field name="specific" help-text="The specific domain of applicability"
/>
  </fields>
</define-compound-classifier>
```