BEA AquaLogic® Enterprise Repository

Configuring and Managing Advanced Registration Flows

Version 3.0
Revised: October 2007

Contents

1. Overview of Advanced Registration Flows

What Are Advanced Registration Flows? 1-2
Example “Community Flow” Use Case. 1-2
Software Components 1-3
ALER Event Manager 1-3
Subscription Manager 1-3
JMS Server. 1-3
Event Monitor 1-3
Advanced Registration Flows. 1-4
Event Management Tools 1-4
Web-based Process Administrator. 1-5
Log Viewer 1-5
Flow Email Notification Templates. 1-5
Flow Configuration Tools. 1-5
Generating a New Config File. 1-5
Refreshing an Existing Config File 1-5
Encrypting Config File Passwords 1-5

2. Getting Started with Advanced Registration Flows

Overview 2-2
Steps to Configure the ALER Event Manager. 2-2
Use Cases 2-3

Configuring the Event Manager 2-3
Triggering an Asset Event 2-4
Steps to Configure and Run the ALBPM Process Engine 2-5
Use Cases 2-6
Configuring the Advanced Registration Flows to Process a Submission Event 2-6
Triggering an Asset Submission Event 2-7

3. Configuring the ALER Event Manager

What Is the ALER Event Manager 3-2
Configuring the Event Manager's System Settings 3-3
Enabling the Event Manager 3-3
Configuring Optional Event Manager Settings 3-3
Eventing Manager Notifier Thread Sleep (seconds) 3-3
Eventing Manager Store Thread Sleep (seconds) 3-4
Eventing Manager Store Delivery Sleep (seconds) 3-4
Batch Size for Event Manager Deliveries 3-4
Configuring the Subscription Manager 3-4
Configuring Web Service Endpoints 3-5
Setting the Expression to Filter Events 3-5
Delivering all Events to an Endpoint 3-5
Delivering Events to an Endpoint Filtered by Event Type 3-6
Delivering Events to an Endpoint Filtered Using a JMS Message Selector 3-6
JMS Message Selector Examples 3-7
Configuring Logging of Event Manager Events 3-8

4. Administrating ALBPM Processes

Overview 4-2
Administering ALBPM Web Applications 4-2

Starting the ALBPM Admin Center 4-2
Starting the ALBPM Process Engine 4-3
Defining the ALBPM Participants 4-4
ALBPM Administrators 4-4
Advanced Registration Flow Participant. 4-4
Tuning the ALBPM Process Engine 4-6
Advanced Properties. 4-6
Database Runtime Properties 4-6
Memory and Execution Thread Properties. 4-6
Configuring a Standalone Process Engine for Failover. 4-7
Using The ALBPM Log Viewer 4-7
Filtering Event Log Messages for ALER Flows 4-7

5. Configuring Advanced Registration Flows

Overview of Advanced Registration Flows. 5-2
Creating and Customizing a Workflow Configuration File 5-2
Generating a Workflow Configuration File 5-2
Defining the ALER Connection and Registrar. 5-3
Encrypting the Registrar User Password 5-3
Wiring Asset Events to Flows 5-4
Automatic Asset Registration Flows 5-6
Configuring Community Flows 5-6
Setting the Community for an ALER Project 5-9
Setting the Community for an Asset Type 5-9
Configuring a Community to Automatically Accept an Asset 5-9
Configuring a Community to Assign Assets for Tab Approval 5-9
Configuring a Community to Assign Assets for Tab Approval Using Multi-tier5-10
Configuring a Community to Automatically Register an Asset 5-10

Configuring a Community to Have a Dedicated Registrar 5-10
Configuring Automated Acceptance and Automated Registration Flows 5-10
Asset Type 5-11
Categorization Settings 5-11
Submitter Role 5-11
Conflict Resolution and Precedence 5-12
Multi-tier Automatic Assignment Flows 5-13
Use Cases 5-13
Using an <alerid> for Tab Approvals 5-14
Setting Up a Community for Multi-tier Tab Approval 5-15
Setting Up an Asset Type for Multi-tier Tab Approval 5-16
Metadata Change Flows 5-18
Use Cases 5-18
Configuring Metadata Change Flows 5-18
Available Metadata Change Events/States 5-18
Available Flows That Can Be Wired to Actions 5-19
Example Metadata Change Configuration 5-20
Example Metadata Change Configuration That Checks for Metadata Value . . 5-21
ChangeClassification 5-21
ChangeCAS 5-21
ChangeAssetLifecycle 5-22
ApproveTabAction 5-22
UnapproveTabAction 5-22
AutoApproveTabAction 5-23
UnapproveChangeManagementTab 5-23
ResetChangeManagementTab 5-24
NotifyCustomUser 5-24
Invoking Flows Based on Approval of Named Tabs 5-24

Time-based Escalation Flows 5-25
Tracking Unsubmitted Assets. 5-26
Tracking Unaccepted Assets 5-27
Tracking Unapproved Assets 5-27
Tracking Unregistered Assets. 5-28
Validation Expiration Flows 5-28
Asset Expiration Warning Notification 5-30
Unregister Assets After Expiration 5-31
Inactivate After Expiration. 5-31
Delete Assets After Expiration. 5-31
Customizing Flow Notification Email Templates. 5-32

6. Configuring JMS Servers for ALER

Overview of JMS for the Event Manager 6-2
Configuring Connectivity Properties for External JMS Servers 6-2
Enabling and Configuring an External JMS Server 6-2
Configuring JMS Message Header Properties 6-3
Miscellaneous JMS Properties 6-4
Configuring External JMS Jar Files 6-4
Configuring the Embedded ActiveMQ JMS Server to Use a Database 6-4
Configuring JMS Durable Subscribers for Web Service Endpoints 6-5
Configuring JMS Servers In an ALER Cluster 6-6
Enabling JMS Clustering Mode 6-6
Configuring Embedded JMS Servers for Clustering 6-6
Configuring External JMS Servers for Clustering 6-7
Configuring a JMS Provider In WebSphere 6.1.0.5 6-8

7. Monitoring and Managing Events

Overview 7-2
Monitoring Events 7-3
Prerequisites 7-3
Usage 7-3
Cleaning Up Stored Events 7-5
Prerequisites 7-5
Usage 7-5
Sample Event Cleanup 7-6
Generating the Workflow Config File 7-6
Refreshing the Workflow Config File 7-7
Encrypting Your Passwords 7-8

8. Extending the Event Manager for Web Service Endpoints

Overview 8-1
Developing a Web Service Endpoint 8-2
Web Service Operations 8-4
Available Web Service Operations 8-4
newEventRequestResponse 8-4
newEventRequestResponseString 8-4
newEventRequest 8-4
newEventRequestString 8-4
newEvent 8-4
Selecting a Web Service Operation 8-5
Developing a Notifier Plug-in 8-6
Developing an Endpoint With an Incompatible Contract 8-7

CHAPTER 1

Overview of Advanced Registration Flows

This section contains information on the following subjects:

- "What Are Advanced Registration Flows?" on page 1-2
- "Example "Community Flow" Use Case" on page 1-2
- "Software Components" on page 1-3

What Are Advanced Registration Flows?

In previous releases of ALER, the asset registration process required the registrar or advanced submitter to manually initiate and monitor the registration process. The required information was gathered and entered on the appropriate tabs in the Asset Editor. The registrar examined each tab and monitored the workflow. When information for a specific stage of the workflow was acceptable, the registrar approved the data on the appropriate tab. The registrar also had the option to edit any of the information for any stage of the process.

The introduction of *Advanced Registration Flows* in ALER 3.0, Advanced Edition, attempts to automate the manual asset registration process by providing a set of predefined flows designed to automate a set of common ALER asset registration tasks, such as asset submission, acceptance, registration, and other governance processes.

To accomplish this ALER 3.0 includes an embedded, JMS-based event engine that manages the flow of ALER asset registration *events*, in the form of Web Service events. These events trigger the pre-defined flows. Once installed, the Advanced Registration Flows can be run out-of-the box or can be tailored to suit your environment.

Note: The flows do not have their own user interface, but will automate certain asset metadata and state changes in the background based on particular ALER events.

For ease of use, you can use the predefined ALPBM endpoint or create your own Web Service endpoints to subscribe to ALER events. There are also event monitoring and logging tools for troubleshooting and tuning purposes.

Example “Community Flow” Use Case

In previous releases, the asset acceptance, assignment, registration processes required multiple registrars to manually initiate and monitor the process from end-to-end via numerous emails. In some cases, there was only one registrar that was notified about the newly submitted assets, and as a result, the registrar could be overloaded with emails about new assets.

The Community flow provides a way to automate the asset acceptance, assignment, and registration process by allowing the configuration of automated assignment rules and also introduces the notion of *federated registrars* among different authorities. Rather than spamming many registrars across all communities (through the system registrar notification), you can limit the system registrar to one or a few individuals, and let the Automatic Acceptance flow accept assets on behalf of a registrar-of-record for the community. The Community flow feature can distribute asset submissions to those with the authority to approve them for the community.

For example, you can add two communities and configure two different registrars responsible for each community. Then, depending on the producing projects or asset types, certain assets can belong to a community. The Community flow automatically accepts such assets in the same way it would be manually accepted by a registrar.

Software Components

Advanced Registration Flows includes the following software components:

ALER Event Manager

The Event Manager emits asset registration events in the form of Web Service messages. These events trigger pre-built flows that automate ALER asset submission, acceptance, registration, and other governance processes. See Chapter 3, “Configuring the ALER Event Manager.”

Subscription Manager

The Subscription Manager is XML-based configuration file that is responsible for managing the event subscriptions by the Web Service endpoints (either the predefined ALPBM endpoint or user-defined endpoints) where matched events will be delivered. The Event Manager uses the `EndPointEventSubscription.xml` file to load information about the endpoints where events need to be delivered. See Chapter 3, “Configuring the ALER Event Manager.”

JMS Server

The Event Manager uses an embedded version of Apache ActiveMQ JMS Server that is enabled by default. The embedded JMS server is configured to run out-of-the-box without any additional configuration. However, you can also configure the Event Manager to use an external JMS server, such as Weblogic Server JMS or IBM MQSeries. See Chapter 6, “Configuring JMS Servers for ALER.”

Event Monitor

A tool to monitor the events that are generated by the Event Manager. The Event Monitor peeks into the event traffic and prints information, such as the event body and event properties. See Chapter 7, “Monitoring and Managing Events.”

Advanced Registration Flows

The Advanced Registration Flows can be run out-of-the box or can be tailored to suit your environment. See Chapter 5, "Configuring Advanced Registration Flows."

- *Community Assignment Flow* – provides a way to automate the asset acceptance, assignment, and registration process by allowing the configuration of automated assignment rules and also provides the notion of *federated registrars* among different authorities. See "Configuring Community Flows" on page 5-6.
- *Automated Acceptance and Automated Registration Flow* – in addition to using the Community Flows to automatically accept and register the assets, a number of user roles can be used to accept and register assets. See "Configuring Automated Acceptance and Automated Registration Flows" on page 5-10.
- *Multi-tier Approval Flow* – structures the tab approval process in multiple steps called *tiers*. Asset approval tabs can be grouped in tiers, and the Mult-tier Approval flow tracks each tier to verify whether all the tabs are approved by the designated approvers. As soon as the last tab in a tier is approved, the flow starts the next tier by assigning the asset to the next level of designated approvers. See "Multi-tier Automatic Assignment Flows" on page 5-13.
- *Metadata Change Flow* – exposes a flexible framework where state changes or metadata changes can be wired to actions. The Metadata Change flows come with the a set of pre-bundled actions. New actions can be developed in the form of ALER flows and can be plugged in. See "Metadata Change Flows" on page 5-18.
- *Time-based Escalation Flow* –track assets in various states and notifies all interested parties. There are four different kinds of Time-based Escalation flows and each one can be configured individually. See "Time-based Escalation Flows" on page 5-25.
- *Validation Expiration Flow* – tracks expired assets prior to the specified expiration date, as well as at the day of expiration, and sends warning notifications to all interested parties. See "Validation Expiration Flows" on page 5-28.

Event Management Tools

There are event monitoring and logging tools for troubleshooting and tuning purposes.

Web-based Process Administrator

The ALBPM Process Execution Administrator actively manages the orchestration of asset registration events in the form of Web Service messages. For more information, see "Administering ALBPM Web Applications."

Log Viewer

The ALBPM Log Viewer enables you to read information logged by the Process Execution Engine. A set of log files is created for each project you define. The Studio Log Viewer reads the files and displays them to help you monitor and trace Engine execution. For more information, see "Using The ALBPM Log Viewer."

Flow Email Notification Templates

The Automated Registration Flows automatically send email notifications under many circumstances. There are five new email templates for the new flows. Administrators can customize the email subject, body, etc., the same way as other email templates. See "Customizing Flow Notification Email Templates".

Flow Configuration Tools

There are flow configuration tools for generating new configuration file, refreshing exisiting files, and encrypting passwords. For more information, see Chapter 7, "Monitoring and Managing Events."

Generating a New Config File

ALER administrators may need to configure and customize flows because there will be new asset types, projects, categorizations, etc. The Generate Config XML tool connects to ALER and creates a new file that can be customized.

Refreshing an Existing Config File

The Refresh Config XML tool lets you to refresh a Config XML file without restarting the Event Manager.

Encrypting Config File Passwords

The security Encrypt Password tool lets you to encrypt the passwords for security reasons.

CHAPTER 2

Getting Started with Advanced Registration Flows

This section contains information on the following subjects:

- "Overview"
- "Steps to Configure the ALER Event Manager"
- "Steps to Configure and Run the ALBPM Process Engine"

Overview

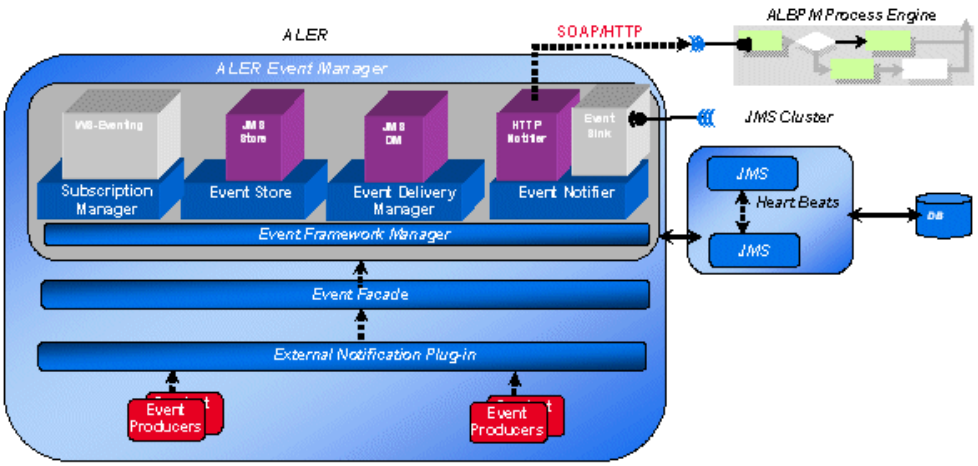
This section will help you to quickly get started using the Advanced Registration Flow feature using the bundled ALPBM Web Service endpoint that is configured to work with the ALBPM Process Engine. However, this feature is highly extensible and can be tailored to suit your environment.

Steps to Configure the ALER Event Manager

The Event Manager is a component embedded within ALER that manages event subscriptions, event persistence, event filtering, and event delivery. Web Service endpoints can subscribe to the Event Manager's Subscription Manager and the asset registration events that are generated within ALER are delivered to the Web Service endpoints.

The following diagram shows the different components that are involved.

Figure 2-1 Advanced Registration Flow Components



The Event Manager uses an embedded version of Apache ActiveMQ JMS Server that is enabled by default. The embedded JMS server is configured to run out-of-the-box without any additional configuration. However, you can also configure the Event Manager to use an external Java-based message broker, such as Weblogic Server JMS or IBM MQSeries.

For more information on configuring the Event Manager, see Chapter 3, "Configuring the ALER Event Manager."

Use Cases

- ALER 3.0 features pre-bundled ALBPM flows and a Web Service endpoint that is by default registered with the Event Manager's Subscription Manager. All the triggered events are delivered to this ALBPM endpoint, which then attempts to automate ALER processes, such as the asset registration process, tracking metadata changes, and taking pre-defined actions.
- You can also write your own Web Service endpoints, subscribe them with the Event Manager, and start getting the events to solve your specific business needs.

Configuring the Event Manager

After ALER is installed, configure the Event Manager as follows.

1. The Event Manager needs to be enabled in ALER to allow the Event Manager to send events to external Web Service endpoints. You can either:
   - Enable the `cmee.eventframework.enabled=true` property in the `eventing.properties` file in the `<ALER Domain>\WEB-INF\classes` directory.

     or

   - This property can also be enabled using the ALER Web-based console's System Settings, as explained in "Configuring the Event Manager's System Settings."
2. The default Eventing `cmee.eventframework.delivery.sleep` and `cmee.eventframework.store.sleep` property values can also be tuned to control the overall performance of ALER and the Web Service endpoints. These properties directly impact the number of events that get triggered per second by the Event Manager. For example, If a faster response is required for testing purposes, the default `cmee.eventframework.store.sleep` value of 7200 seconds should be changed to a reasonable testing amount.
3. The Event Manager uses the same logging framework as ALER. By default, logging is enabled to go to a file, but you direct the debug statements to go to the console by appending the following categories to the `log4fl.properties` file in the `<ALER Domain>\WEB-INF\classes` directory.

```
# eventing subsystem
log4j.category.com.bea.infra.event.core= debug,eventingLog,stdout
log4j.category.com.bea.infra.event.dm= debug,eventingLog,stdout
log4j.category.com.bea.infra.event.facade= debug,eventingLog,stdout
log4j.category.com.bea.infra.event.notifier= debug,eventingLog,stdout
```

```
log4j.category.com.bea.infra.event.store= debug,eventingLog,stdout
log4j.category.com.bea.infra.event.sub= debug,eventingLog,stdout
```

4. Configure the Web Service subscriptions with the Event Manager's Subscription Manager.

   Note: By default the Subscription Manager is configured to work out-of-the-box with the ALBPM Process Engine *if* the ALBPM Process Engine is running on the same machine as ALER. You can skip this step if this is the case because the default settings are ready to run.

   As shown below, the following information may need to be changed within the `EndPointEventSubscription.xml` file under `<ALER Domain>\WEB-INF\classes` directory, depending on the requirement:

   – **Host** – If the Web Service Endpoint is running in a host other than ALER. If it'' the same host, leave the default unchanged

   – **Port** – Specify the port of the Web Service Endpoint. If ALBPM is used as the Process Engine, leave the default unchanged.

   – **URI** – Specify the URI of the Web Service. If ALBPM is used as the Process Engine, leave the default unchanged.

   – **Operation Name** – If ALBPM is used as the Process Engine, leave the default unchanged. Please refer to the WSDL within the `eventNotifier.jar` located in `<aler Webapp path>/WEB-INF/lib` for the available operations.

   – **User Name/Password** – Used only if ALBPM is used as the Process Engine. Default user name and password are "admin" and "admin".

   – **Expression** – Default is empty, which means all the events are delivered.

5. Restart ALER for the configuration changes to take effect.

Triggering an Asset Event

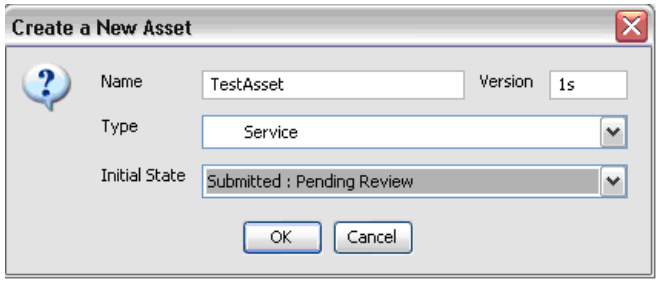
Follow these steps to make sure that events are triggered after the configuring the Event Manager.

1. Launch the ALER Asset Editor from the Web-based console.

   For information on using the ALER Asset Editor, refer to the *ALER Registrar Guide*.

2. Create a new asset, as shown here.

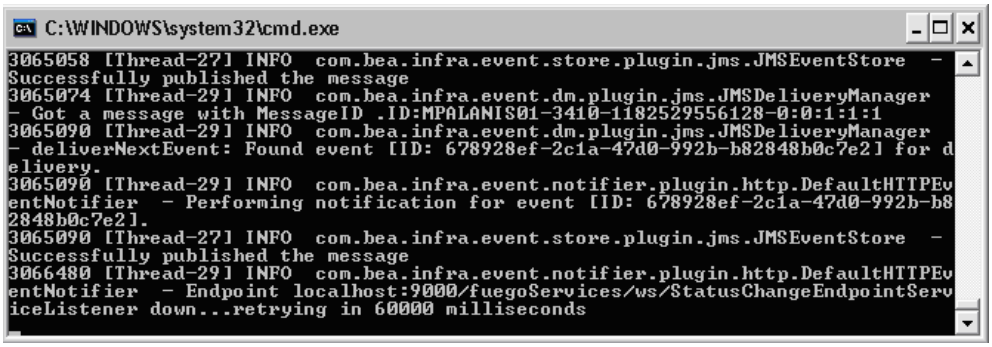
Figure 2-2 ALER Asset Editor - Create New Asset



Note: The Asset Type should be *Service*.

3. Click **OK** to submit the asset.
4. After the asset is submitted, switch to the ALER console to verify the following logging statements printed to the console.

Figure 2-3 Event Monitoring Console



5. The Event Monitoring tool can be used to view the payload of the event that will be delivered. For more information about monitoring events, see "Monitoring Events."

Steps to Configure and Run the ALBPM Process Engine

After the Event Manager is ready to send events, the ALBPM Process Engine needs to be configured and be ready to process the events. When ALER is installed, it provides an option to install and configure the Process Engine. This section assumes that the Process Engine was successfully installed before following these steps.

To launch the ALBPM Admin Center, double-click the **albpmadmcenter.exe** file in the *<ALBPM Enterprise Home>*\bin directory.

Use Cases

ALER features pre-bundled Advanced Registration Flows that are deployed to the Process Engine. When events are triggered within ALER, they are delivered to the Process Engine and execute the Advanced Registration Flows that attempt to automate ALER processes, such as asset submission, acceptance, registration, and other governance process.

For more information about the available Advanced Registration Flows, see Chapter 5, "Configuring Advanced Registration Flows."

Configuring the Advanced Registration Flows to Process a Submission Event

Follow these steps after the ALBPM Process Engine is installed.

1. Generate the Workflow Configuration (`workflow.xml`) file using the Generate Workflow Config tool (`config_gen.bat`). This tool connects to ALER and creates a bootstrapping file that can be customized. For more information about generating the `workflow.xml` file, see "Generating the Workflow Config File."
2. Copy the newly generated `workflow.xml` file to the *<ALBPM Enterprise Edition>*`/enterprise/server/aler_engine` directory.
3. Open the `workflow.xml` file using the XML editor of choice.
4. Make sure that the ALER Connection information, such as the URI and the registrar user name/password, are configured correctly as shown here.

```
<alerconnection>

<uri>http://server01.amer.bea.com:7005/aler/services/FlashlineRegistry
</uri>
     <registrar>
       <user>admin</user>
       <password>admin</password>
     </registrar>
   </alerconnection>
```

The URI must use the following format:
`http://<host>:<port>/<aler web app name>/services/FlashlineRegistry`

5. Within the `workflow.xml` file, locate the `assetType` settings for the "Service" asset type, as shown here.

```
<assetType name="Service" community="_CHANGE_COMMUNITY_" id="154">
  <allTabs>
  <allTabs>
    <tab name="Oveview"/>
    <tab name="UDDI: Business Entity"/>
    <tab name="Taxonomy"/>
    <tab name="Architecture"/>
  </allTabs>
```

6. Add the **autoAccept** attribute and set the value to **true**, as shown here.

```
  <assetType name="Application" community="_CHANGE_COMMUNITY_" id="154"
autoAccept="true">
    <allTabs>
    <allTabs>
      <tab name="Oveview"/>
      <tab name="UDDI: Business Entity"/>
      <tab name="Taxonomy"/>
      <tab name="Architecture"/>
    </allTabs>
```

   Now the ALBPM Process Engine is configured to automatically accept any asset of type "Service."

7. If the ALBPM Process Engine is running, stop it and then restart it to load the latest `workflow.xml` changes.

8. The Refresh Workflow Configuration tool can be used to refresh the `workflow.xml` file without restarting the ALBPM Process Engine. For more information about refreshing the `workflow.xml` file, see "Refreshing the Workflow Config File."

Triggering an Asset Submission Event

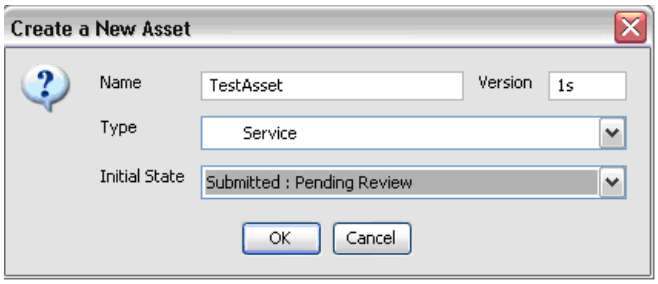
Once the ALBPM Process Engine is configured and running, follow these steps to process an asset submission event.

1. Launch the ALER Asset Editor from the Web console.

   For information on using the ALER Asset Editor, refer to the ALER Registrar Guide.

2. Create a new asset from File ->New as shown below.

Figure 2-4 ALER Asset Editor - Create New Asset



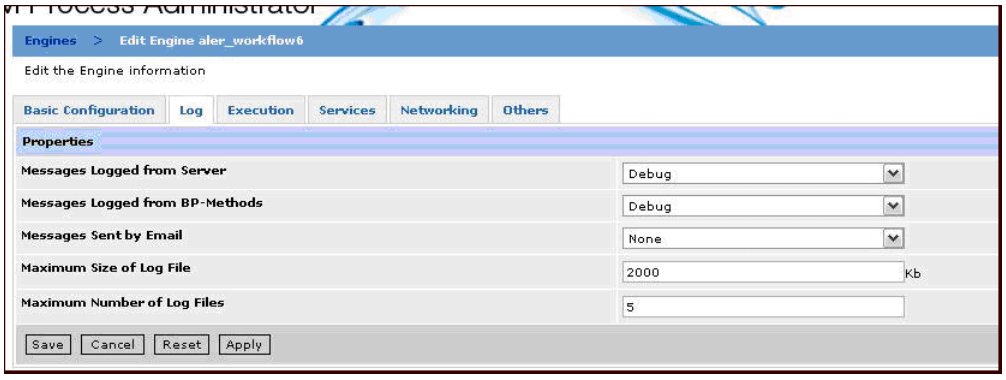
Note: The Asset Type should be *Service*.

3. Click **OK** to submit the asset.

4. After the asset is submitted, switch to the ALBPM Log Viewer to make sure that the event is processed. To launch the Log Viewer, double-click the **albpmlogviewer.exe** file in the *<ALBPM Enterprise Home>*\bin directory.

5. Turn on the "Debug" level on the Log page of the Process Engine using the Process Administrator preference settings. By default, the level is set to "Warning."

Figure 2-5 ALBPM Process Administrator - Logging Preferences



6. When you turn on the Debug level though you will notice that the Process Engine prints a lot of information, not just for the ALER Advanced Registration Flows, but other Process Engine information as well. To filter the ALER logging, follow these steps:

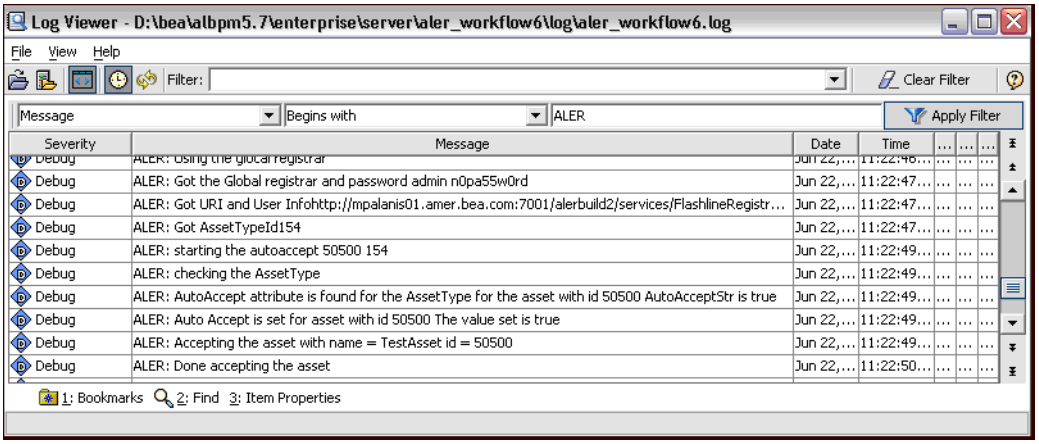
   a. Within the Log viewer, select **Message** in the left-most list box.

   b. Select **Begins With** in the next list box.

   c. Type **ALER:** in the text box

   d. Click the **Apply Filter** button.

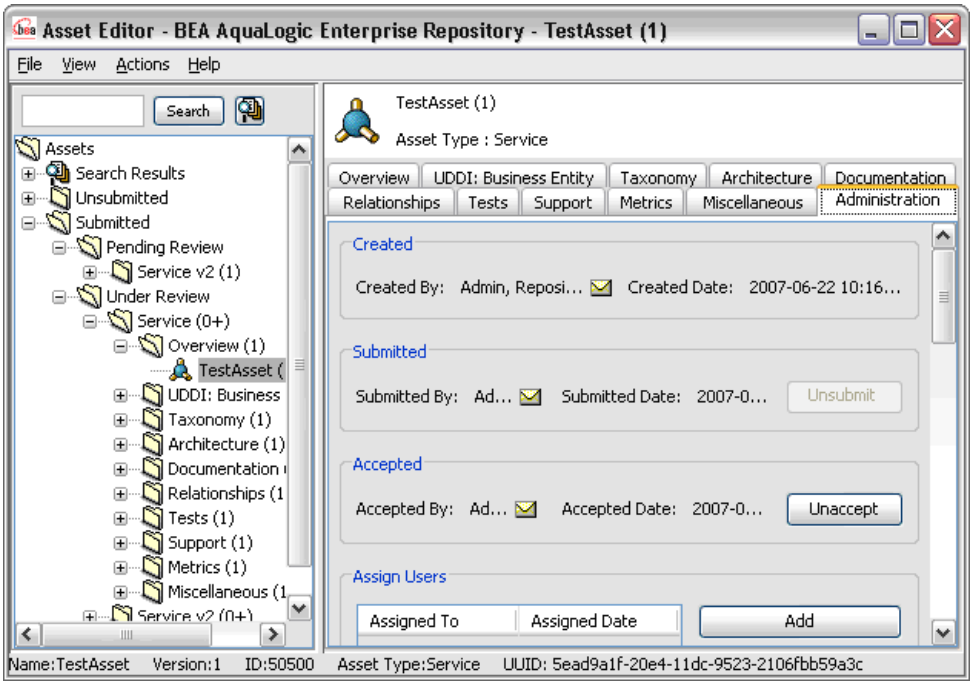
Figure 2-6 ALBPM Log Viewer



7. After the “ALER: Done accepting the asset” message is displayed in the Log Viewer, switch back to the Asset Editor, and then refresh the Administration tab using the **View -> Refresh Tree** command

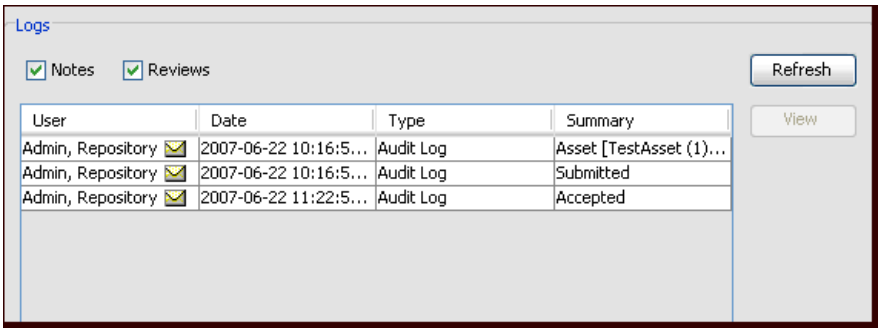
8. Verify that the “Accepted” section is updated with the latest data, as shown here.

Figure 2-7 ALER Asset Editor - Administration Tab



9. Also verify that the Audit Log on the Administration tab is updated, as shown here.

Figure 2-8 ALER Asset Editor - Audit Log



CHAPTER 3

Configuring the ALER Event Manager

This section discusses the Event Manager configuration that needs to be completed before using the Advanced Registration Flows. It contains information on the following subjects:

- "What Is the ALER Event Manager" on page 3-2
- "Configuring the Event Manager's System Settings" on page 3-3
- "Configuring the Subscription Manager" on page 3-4
- "Configuring Logging of Event Manager Events" on page 3-8

What Is the ALER Event Manager

The Event Manager is a component embedded within ALER that manages event subscriptions, event persistence, event filtering, and event delivery. Web Service endpoints can subscribe to the Event Manager's Subscription Manager and the asset registration events that are generated within ALER are delivered to the Web Service endpoints.

The following diagram shows the different components that are involved.

Figure 3-1 Advanced Registration Flow Components

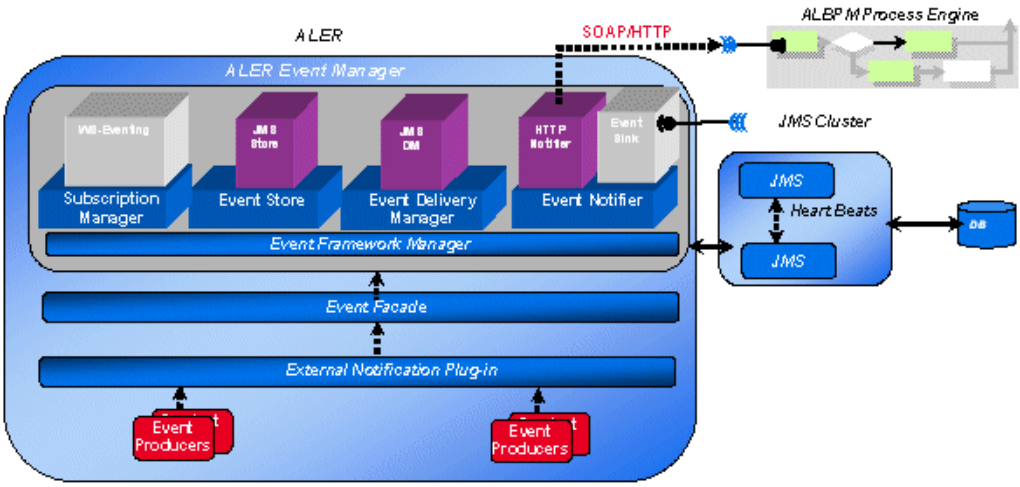


Figure 3-2 Advanced Registration Flow Components

The Event Manager uses an embedded version of Apache ActiveMQ JMS Server that is enabled by default. The embedded JMS server is configured to run out-of-the-box without any additional configuration. However, you can also configure the Event Manager to use an external JMS server, such as Weblogic Server JMS or IBM WebSphere.

This section discusses the Event Manager configuration that needs to be completed before using the Advanced Registration Flows. For information on configuring the Advanced Registration Flows, see Chapter 5, "Configuring Advanced Registration Flows."

Configuring the Event Manager's System Settings

ALER's System Settings section allows administrators to configure the basic ALER operation and to enable/disable specific features. The Event Manager-related settings are under the "Eventing" group under the main "External Integrations" category. For more information about System Settings, see the *ALER Administration Guide*.

Additional "Eventing" properties are available for configuring an external JMS server, such WebLogic Server and IMB WebSphere, and are described in Chapter 6, "Configuring JMS Servers for ALER."

Enabling the Event Manager

The Event Manager needs to be enabled in ALER to allow the Event Manager to send events to external Web Service endpoints.

1. Click **System Settings** in the sidebar on the ALER **Admin** screen.
2. Enter **Event** in the System Settings Search box to view all the Event Manager related settings.
3. Click **True** next to the **Enable Event Manager** property.
4. Click **Save**.
5. Restart ALER for the configuration changes to take effect.

Configuring Optional Event Manager Settings

There are some optional "Eventing" properties that you can use to tune the Event Manager performance.

Note: You must restart ALER after changing any Eventing property in order for the changes to take effect.

Eventing Manager Notifier Thread Sleep (seconds)

If an endpoint is not unavailable when one or more events should be delivered to that endpoint, the Event Manager notifier will retry delivering the event until the endpoint is available. The `cmee.eventframework.notifier.sleep` property configures in seconds how long the notifier should wait before trying to redeliver an event.

Eventing Manager Store Thread Sleep (seconds)

As soon as an event is triggered, the Event Manager stores the event in memory before pushing it to the JMS server so that the ALER thread is not blocked. The `cmee.eventframework.store.sleep` property specifies how long the Event Manager's Store Manager thread should sleep before polling for the next event stored in memory. The default polling delay is 60 seconds.

Eventing Manager Store Delivery Sleep (seconds)

By default, the Event Manager delivers events in batches. The `cmee.eventframework.delivery.sleep` property specifies how long the Event Manager's Delivery Manager thread should sleep before selecting the next available batch of events from the JMS server. The default delay between each batch is 7200 seconds (two hours).

Tip: The default `cmee.eventframework.store.sleep` and `cmee.eventframework.delivery.sleep` property values can be tuned to control the overall performance of ALER and the Web Service endpoints. These properties directly impact the number of events that get triggered per second by the Event Manager. For example, If a faster response is required for testing purposes, the default `cmee.eventframework.delivery.sleep` value of 7200 seconds should be changed to a reasonable testing amount.

Batch Size for Event Manager Deliveries

When the Event Manager delivers events in batches, the delivered batch size can be configured using the `cmee.eventframework.delivery.batch.size` property. The default batch size is 100 events. If the Event Manager finds less number of events to deliver, it will deliver the available events and then sleep for the time configured in the `cmee.eventframework.delivery.sleep` property.

Configuring the Subscription Manager

The Subscription Manager is responsible for managing the event subscriptions by the Web Service endpoints where the matched events will be delivered.

The Subscription Manager configuration file is located in `<aler webapp name>\WEB-INF\classes\EndPointEventSubscription.xml`.

Configuring Web Service Endpoints

The Event Manager uses the `EndPointEventSubscription.xml` file to load information about the Web Service endpoints where events need to be delivered. The host, port, URI, user and password of the predefined ALPBM endpoint, or user-defined Web Service endpoint, need to be configured, as shown in this example snippet:

```
<sub:EventSubscriptionData
xmlns:sub="http://www.bea.com/infra/events/subscription" xmlns:xsi=???
  <sub:eventSubscription>
    <sub:endPoint name="ALBPMEndpoint">
      <sub:host>maplanis01.amer.bea.com</sub:host>
      <sub:port>9000</sub:port>
      <sub:uri>fuegoServices/ws/StatusChangeEnpointServiceListener</sub:uri>
      <sub:targetNamespace>StatusChangeEndpoint</sub:targetNamespace>
      <sub:operationName>newEvent</sub:operationName>
      <sub:authenticationData>
        <sub:basicAuthentication>
          <sub:username>aler_workflow_user</sub:username>
          <sub:username>aler_workflow_user</sub:username>
        </sub:basicAuthentication>
      </sub:authenticationData>
    </sub:endPoint>
  <sub:notifierClass>com.bea.infra.event.notifier.help.AlbpmHTTPEventNotifier
</sub:notifierClass>
    <sub:expression>id &gt; 500</sub:expression>
  </sub:eventSubscription>
</sub:EventSubscriptionData>
```

As many endpoints can be added as desired and the endpoints can be located in different hosts or ports and the events can be load balanced. The pre-defined Advanced Registration Flow has just one endpoint called "`StatusChangeEndpoint`".

Setting the Expression to Filter Events

Events can be filtered based on the value entered in the `expression` element.

Delivering all Events to an Endpoint

The default setting is to deliver *all events* to an endpoint. All the events that are triggered within ALER are delivered to the OOTB endpoint when the `expression` element is empty.

```
<sub:expression></sub:expression>
```

Delivering Events to an Endpoint Filtered by Event Type

The following XML snippet shows how to deliver an event of type `AssetSubmission` to an endpoint:

```
<sub:expression> eventdata_name
=”urn:com:bea.aler:events:type:AssetSubmission”</sub:expression>
```

You can also use the “`OR`” operator to filter more than one event type:

```
eventdata_name =”urn:com:bea.aler:events:type:AssetSubmission” OR
eventdata_name =”urn:com:bea.aler:events:type:AssetAccepted”
```

These are the event types that are supported:

- `urn:com:bea:aler:events:type:AssetSubmission`
- `urn:com:bea:aler:events:type:AssetAccepted`
- `urn:com:bea:aler:events:type:AssetTabApproved`
- `urn:com:bea:aler:events:type:AssetAllTabApproved`
- `urn:com:bea:aler:events:type:AssetRegister`
- `urn:com:bea:aler:events:type:PolicyAssertionChanged`
- `urn:com:bea:aler:events:type:MetaDataChange:name`
- `urn:com:bea:aler:events:type:AssetUnSubmission`
- `urn:com:bea:aler:events:type:AssetUnAccept`
- `urn:com:bea:aler:events:type:AssetUnregister`
- `urn:com:bea:aler:events:type:AssetStatusChanged`
- `urn:com:bea:aler:events:type:MetaDataChange:version`
- `urn:com:bea:aler:events:type:MetaDataChange:description`
- `urn:com:bea:aler:events:type:CategorizationChanged:assetLifecycleStage`
- `urn:com:bea:aler:events:type:CategorizationChanged:classification`
- `urn:com:bea:aler:events:type:MetaDataChange:supported`
- `urn:com:bea:aler:events:type:MetaDataChange:organizational ownership`
- `urn:com:bea:aler:events:type:MetaDataChange:usagefee`

Delivering Events to an Endpoint Filtered Using a JMS Message Selector

Selectors are a way of attaching a filter to a subscription to perform content-based routing. Selectors are defined using SQL 92 syntax. The following is a complete list of fields that can be

used to write a filter expression to filter the events. These fields are added to the JMS message as properties by the Event Manager and a JMS Message Selector that accesses the fields can be written to filter the events.

```
submittedby_emailaddress = mrsmith@bea.com
asset_description = Test Asset
submittedby_name = aler_workflow_user
submittedby_id = 99
asset_community = Java
eventdata_description = new aler event
eventsource_componentname = Aqualogic ALER
asset_name = TestAsset
eventsource_componenttype = ALER3.0
asset_typeid = 154
eventdata_eventid = d0cdac55-c78f-4a29-8aec-6ea9ba8d31f1
eventdata_name = urn:com:bea:aler:events:type:MetaDataChange:name
asset_activestatus = ACTIVE
eventsource_location = ALERCore
asset_id = 50100
eventdata_version = ver1.0
asset_version = 1
```

For more information about JMS Message Selectors, refer to the following web sites:

- http://java.sun.com/j2ee/1.4/docs/api/javax/jms/Message.html
- http://activemq.apache.org/selectors.html

JMS Message Selector Examples

Here are some sample usages of JMS message selectors:

- `asset_id BETWEEN 50000 AND 50100`
- `eventdata_name = 'urn:com:bea:aler:events:type:AssetSubmission' AND asset_id BETWEEN 50000 AND 50100`
- `asset_name LIKE 'Inventory'`
- `asset_id > 500`

Tip: Symbols, such as “< >” used for less than/greater than, are not valid XML content. This is because the expression is written in an XML file and parsed by the Event Manager, the

XML unfriendly characters need to be mangled using the XML Rules. For example, you must use "`id > 500`", which is equivalent to "`asset_id > 500`".

Configuring Logging of Event Manager Events

The Event Manager uses the same logging framework as ALER. By default, logging is enabled to go to a file, but you direct the debug statements to go to the console by appending the following categories to the `log4fl.properties` file in the `<ALER Domain>\WEB-INF\classes` directory.

```
# eventing subsystem
log4j.category.com.bea.infra.event.core= debug,eventingLog,stdout
log4j.category.com.bea.infra.event.dm= debug,eventingLog,stdout
log4j.category.com.bea.infra.event.facade= debug,eventingLog,stdout
log4j.category.com.bea.infra.event.notifier= debug,eventingLog,stdout
log4j.category.com.bea.infra.event.store= debug,eventingLog,stdout
log4j.category.com.bea.infra.event.sub= debug,eventingLog,stdout
```

CHAPTER 4

Administrating ALBPM Processes

This section contains information on the following subjects:

- Overview
- Administering ALBPM Web Applications
- Tuning the ALBPM Process Engine
- Configuring a Standalone Process Engine for Failover
- Using The ALBPM Log Viewer

Overview

After the Event Manager is ready to send events, the Process Engine needs to be configured in order to be ready to process the Events. When ALER is installed, it provides an option to install and configure the ALBPM Process Engine. This section assumes that the ALBPM Process Engine was successfully installed.

Administering ALBPM Web Applications

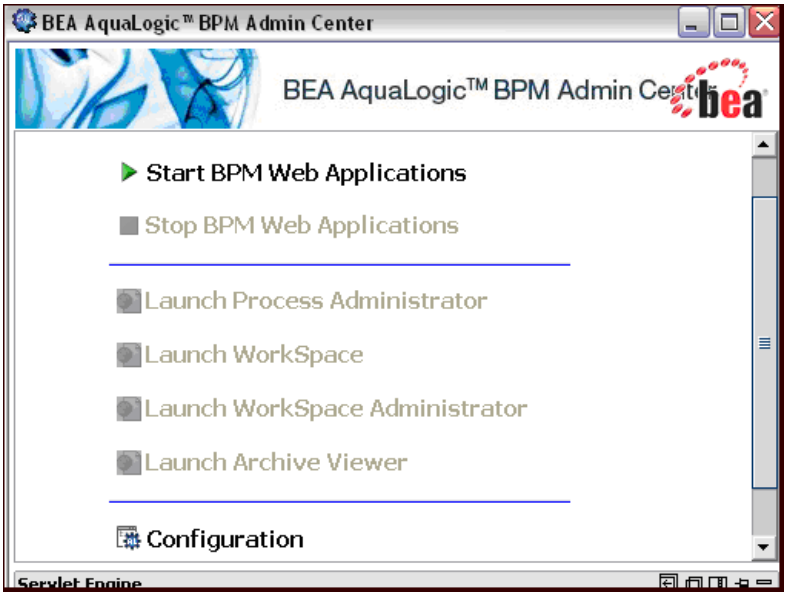
To start the ALBPM Process engine and define the participants, you must launch the ALBPM Admin Center.

Starting the ALBPM Admin Center

Follow these steps to launch the ALBPM Admin Center:

1. Navigate to the `<BEA_HOME>\albpm57\enterprise\bin` directory and double-click one of the following files:
   - **albpmadmcenter.exe** (Windows or UNIX GUI-based)
   - **./startwebconsole.sh** (UNIX console-based). Then point your browser to `http://<host>:8585/webconsole` (e.g., `http://localhost:8585/webconsole`).
2. On the Admin Center page, click the **Start BPM Web Applications** option.

Figure 4-1 ALBPM Admin Center



3. When it becomes available, click the **Launch Process Administrator** option to launch the Process Administrator.

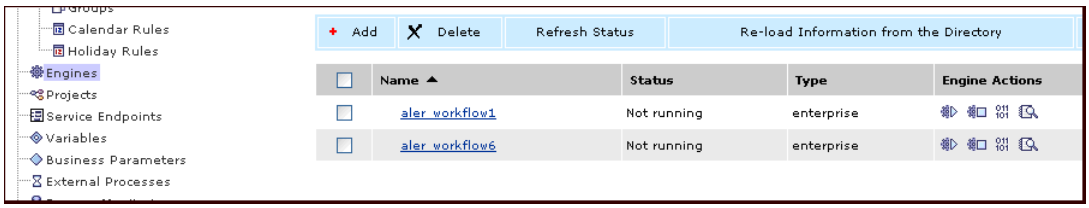
4. When prompted to enter the required credentials, enter the BPM admin user name and password that was used on the FDI User Credentials panel during the installation process. The recommended example for these credentials is `bpm_admin` for the user name and password.

Starting the ALBPM Process Engine

Follow these steps to start the ALBPM Process Engine.

1. On the ALBPM Process Administrator page, open the `aler_engine` Process Engine by clicking the **Engine** link on the left side of the page.

Figure 4-2 ALBPM Process Administrator - Start / Stop



2. Start the `aler_engine` by clicking the **Start** icon under Engine Actions on the right side of the page. Starting the engine may take several minutes to complete. Make sure that the status of the engine is `Ready`.

Once you the ALBPM Process Engine is running, you can stop it and then restart it to load your latest `workflow.xml` changes.

Defining the ALBPM Participants

This section explains how to define the ALPBM Process Engine participants.

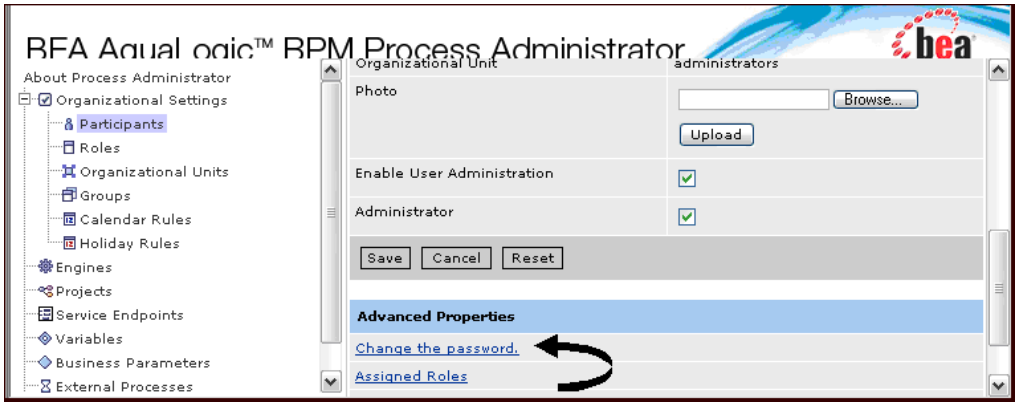
ALBPM Administrators

Using the FDI User Credentials, ALBPM Process Administrator can log into the Process Administrator, start/stop the process engine, and create other users.

Advanced Registration Flow Participant

When the ALBPM Process Engine is installed by the BEA Products installer, it creates `aler_workflow_user` as the Advanced Registration Flow user. By default, the password is also set as `aler_workflow_user`, but the password can be changed in the Process Administrator, as shown here.

Figure 4-3 ALBPM Process Administrator - Change Password



A new participant can also be created for the role of “administrator” and this new participant can be configured in the Event Manager’s Subscription Manager file. For more information, see “Configuring the Subscription Manager”.

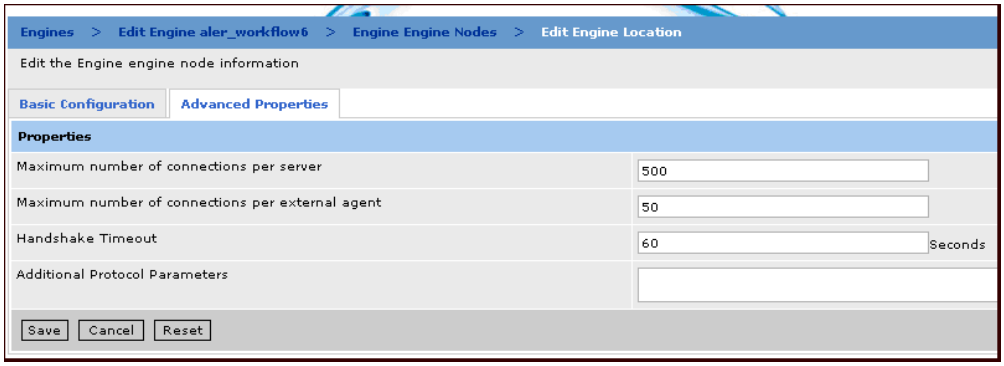
Tuning the ALBPM Process Engine

The following parameters need to be tuned using the ALBPM Process Administrator.

Advanced Properties

Go to the **Engines** > <***Engine Name***> > **Engine Nodes** > **Advanced Properties** page.

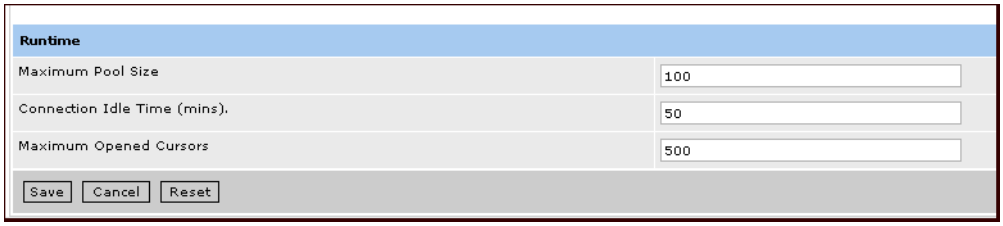
Figure 4-4 ALBPM Process Administrator - Advanced Properties



Database Runtime Properties

Go to the **Engines** > <***Engine Name***> > **Edit Engine Database Configuration** page.

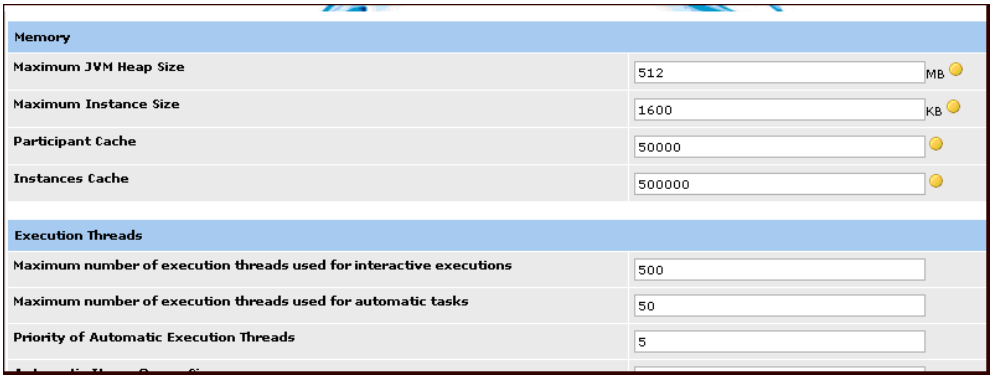
Figure 4-5 ALBPM Process Administrator - Database Runtime



Memory and Execution Thread Properties

Go to the **Engines** > <***Engine Name***> > **Execution** page.

Figure 4-6 ALBPM Process Administrator - Memory and Threads



Configuring a Standalone Process Engine for Failover

To support failover of ALBPM standalone process engines, you can configure a backup engine(s) in your environment. One of the engines in this federation is marked as **PRIMARY** and the others assume to be backups for this primary engine. Multiple engines can be configured to serve as backups. Any of these backup engines will take the role of the primary if the designated primary fails. When the server that has failed comes back online, it will join in as a backup to the one acting as primary.

For detailed instructions on configuring backup engines, see `http://edocs.bea.com/albsi/docs55/pdfs/Fuego5-EngineFailover.pdf`.

Using The ALBPM Log Viewer

The ALBPM Log Viewer enables you to read information logged by the Process Execution Engine. A set of log files is created for each project you define. The Studio Log Viewer reads the files and displays them to help you monitor and trace Engine execution.

To launch the Log Viewer, double-click the **albpmlogviewer.exe** file in the *<ALBPM Enterprise Home>*`\bin` directory.

Filtering Event Log Messages for ALER Flows

You can filter log messages so that the Advanced Registration Flows log Info, Debug, and Fatal messages.

Turn on the “Debug” level on the Log page of the Process Engine using the Process Administrator preference settings. By default, the level is set to “Warning”.

Go to the **Engines** > <***Engine Name***> > **Log** page.

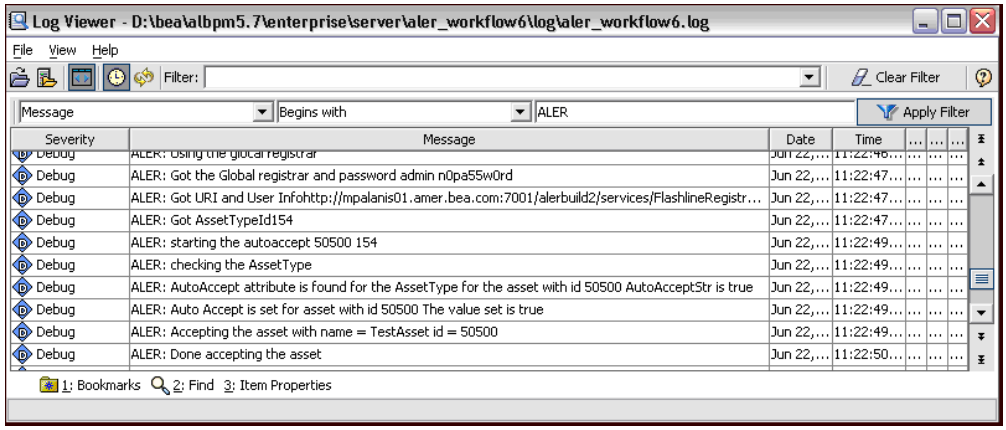
Figure 4-7 ALBPM Process Administrator - Logging Preferences

When you turn on the Debug level though you will notice that the Process Engine prints a lot of information, not just for the ALER Advanced Registration Flows, but other Process Engine information as well. To filter the debug logging to show only the ALER flow-related information, follow these steps:

1. Within the Log viewer, select **Message** in the left-most list box.
2. Select **Begins With** in the next list box.
3. Type **ALER:** in the text box
4. Click the **Apply Filter** button.

The ALER Event Logging prints a prefix of `ALER:` for all logged event messages, as shown here.

Figure 4-8 Log Viewer With ALER Filter



CHAPTER 5

Configuring Advanced Registration Flows

This section contains information on the following subjects:

- Overview of Advanced Registration Flows
- Creating and Customizing a Workflow Configuration File
- Wiring Asset Events to Flows
- Automatic Asset Registration Flows
- Multi-tier Automatic Assignment Flows
- Metadata Change Flows
- Time-based Escalation Flows
- Validation Expiration Flows
- Customizing Flow Notification Email Templates

Overview of Advanced Registration Flows

Tip: Before you begin, you should read Chapter 2, "Getting Started with Advanced Registration Flows" to quickly get started using the Advanced Registration Flow feature using the bundled ALPBM Web Service endpoint that is configured to work with the ALBPM Process Engine.

ALER bundles pre-built ALBPM flows that attempt to automate ALER asset submission, acceptance, registration and other governance process. This section discusses the configuration that is required before starting the ALBPM Process Engine to process the asset events that are triggered by ALER. For more information about configuring the Process Engine to trigger flows, see Chapter 3, "Configuring the ALER Event Manager."

The flows are also designed to be flexible and can be customized using the Workflow Configuration file (`workflow.xml`). This section also discusses each flow in detail and gives examples of how to tailored to suit your environment.

Creating and Customizing a Workflow Configuration File

This section explains how to create and customize a Workflow Configuration XML file.

Generating a Workflow Configuration File

Generate the `workflow.xml` file using the Generate Workflow Config tool (`config_gen.bat`). This tool connects to ALER and creates a bootstrapping file that can be customized. For more information about generating the `workflow.xml` file, see "Generating the Workflow Config File."

1. From a command prompt, run the Generate Workflow Config tool as follows:

   ```
   > config_gen.bat URI User Password ConfigDir
   ```

   where:

   – `URI` = ALER URI, using the following format:
   `http://<host>:<port>/<aler web app name>/services/FlashlineRegistry`
   For example: http://localhost:7001/alerbuild/services/FlashlineRegistry

   – `User` = ALER user name

   – `Password` = ALER password

– `ConfigDir` = the directory where the `workflow.xml` file will be created

Note: If a file already exists, it will be renamed to `workflow.xml.bak`.

2. Copy the newly generated `workflow.xml` file to the *<ALBPM Enterprise Edition>*`/enterprise/server/aler_engine` directory.
3. Open the `workflow.xml` file using the XML editor of choice.

Defining the ALER Connection and Registrar

The Workflow Configuration file will load the ALER connection and registrar information from the following XML data.

```
<alerconnection>
  <uri>http://localhost.7001/aler/services/FlashlineRegistry</uri>
  <registrar>
    <user>admin</user>
    <password>n0pa55w0rd</password>
  </registrar>
</alerconnection>
```

Encrypting the Registrar User Password

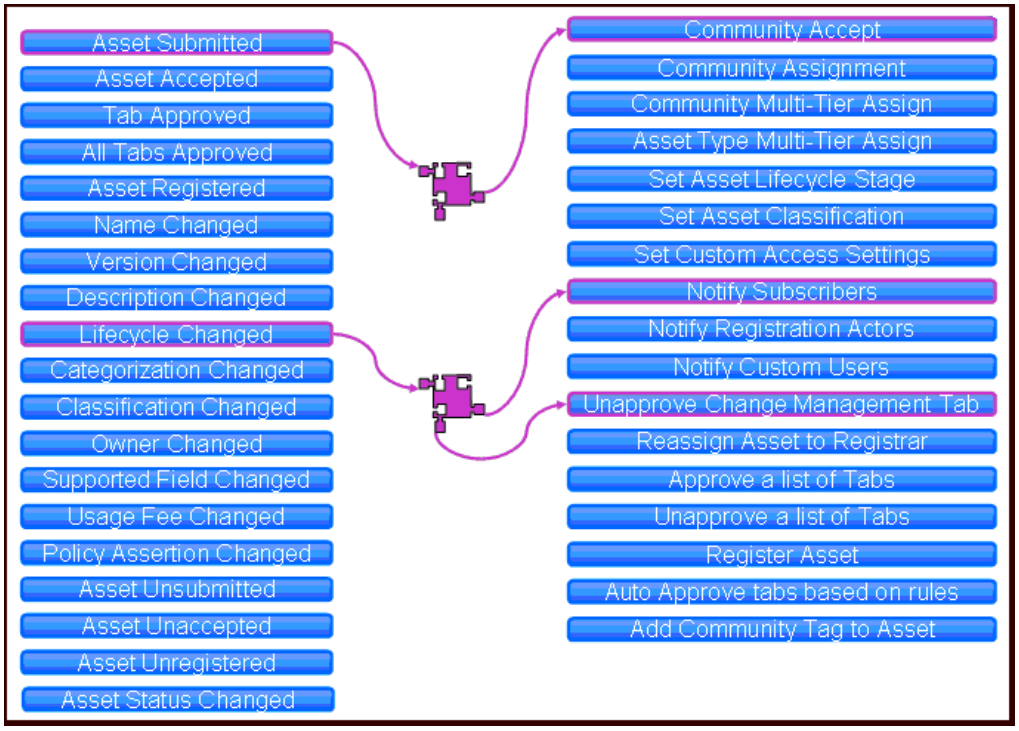
The Security Encrypt Password tool (`runWfSecurity.bat`) allows you to encrypt the registrar passwords that are stored in the Workflow Config file. The tool recursively scans the file and encrypts all the password elements it encounters.

For more information see "Encrypting Your Passwords."

Wiring Asset Events to Flows

The Advanced Registration Flows are designed with a flexible framework where asset events can be wired to one or more flows that will be executed when an event is triggered, as illustrated in Figure 5-1.

Figure 5-1 Wiring Asset Events to Flows



Note: All the events are wired to pre-defined flows out-of-the-box. The wirings only need to be changed if customizations or new flows are designed.

The wiring of asset events to flows is configured within the Workflow Configuration file. For example, the following configuration snippet shows that when an “Asset Submitted” event is triggered, it in turn triggers a flow to automatically accept the asset based on rules that are configured in the Workflow Configuration file.

```
<!--Community Flows-->
<state name="urn:com:bea:aler:events:type:AssetSubmission">
```

```
  <action>CommunityAccept</action>
</state>

<!--The Multi_tier Flows-->
<state name="urn:com:bea:aler:events:type:AssetAccepted">
  <action>MultiTier_Tier1_Assign</action>
</state>
<state name="urn:com:bea:aler:events:type:AssetTabApproved">
  <action>MultiTier_NextTier_Assign</action>
</state>

<!--Asset Registration Status Flows-->
<state name="urn:com:bea:aler:events:type:AssetAllTabApproved">
  <action>AllTabApproved_Register</action>
</state>
```

This example configuration wires the following events to various flows. The `<action>` element contains the name of the flow that will be executed.

1. When an asset “submitted” event is triggered, execute the Community Accept flow.
2. When an asset “accepted” event is triggered, execute the MultiTier1 flow.
3. When a tab “approved” event is triggered, execute the Multi-Tier Next Tier flow.
4. When “all the tabs approved” event is triggered, execute the Automatic Registration flow.

Some of the flows take parameters that are needed as input. Different parameters are passed to different flows. For example, the ChangeCAS (Change Custom Access Settings) flow takes `<customAccessSettings>` as a parameter. Here is a sample wiring when an asset is registered, where the flow automatically assigns `MyCAS` and `MyCAS2` custom access settings.

```
<state name="urn:com:bea:aler:events:type:AssetRegister">
  <action>ChangeCAS</action>
<customAccessSettings>
  <customAccessSetting>MyCAS</customAccessSetting>
  <customAccessSetting>MyCAS2</customAccessSetting>
</customAccessSettings>
</state>
```

Automatic Asset Registration Flows

This section describes how the Advanced Registration flows can automate the manual asset acceptance and registration process done using the ALER Asset Editor. For information on using the ALER Asset Editor and the asset registration process, refer to the *ALER Registrar Guide*.

Note: Do not enable the "Community Acceptance" or the "Automated Acceptance" flows if repository users submit assets via the "Submit an Asset" link. This configuration is not currently supported in ALER.

Configuring Community Flows

The Community flow provides a way to automate the asset acceptance, assignment, and registration process by allowing the configuration of automated assignment rules and also introduces the notion of *federated registrars* among different authorities. Rather than spamming many registrars across all communities (through the system registrar notification), you could limit the system registrar to one or a few individuals, and let the Automatic Acceptance flow accept assets on behalf of a registrar-of-record for the community. The Community flow feature can distribute asset submissions to those with the authority to approve them for the community.

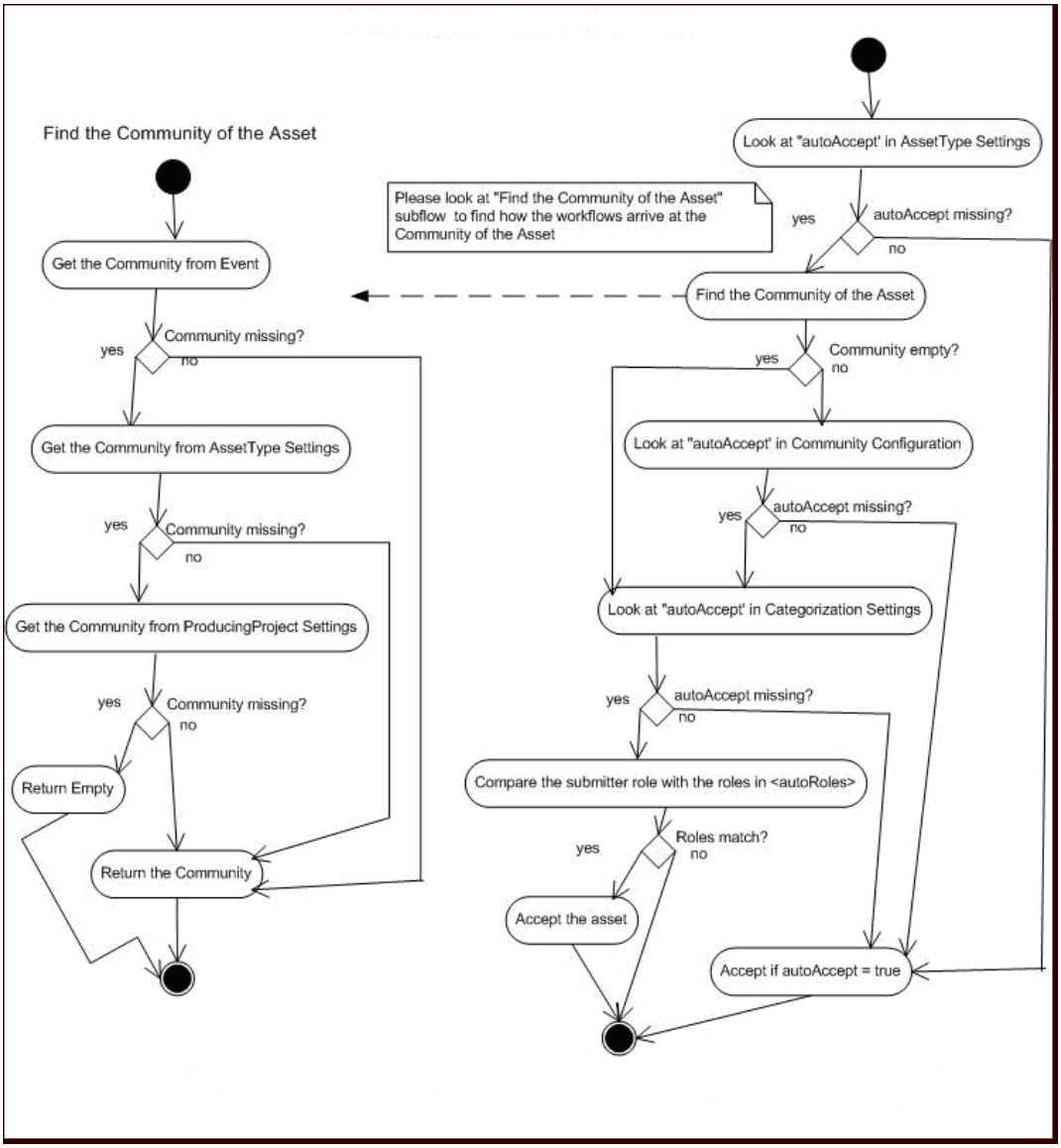
The Community flow can be used to address the following scenarios:

- Automatic federated registrars support for acceptance as opposed to a single registrar getting many notifications about newly submitted assets.
- Even if asset acceptance is manual, the Community flow can be used to automate the assignment of the asset approvals to pre-defined approvers. Creating pre-defined approvers can be achieved in two ways:
  - Creating a list of pre-defined approvers for all the tabs in that asset.
  - Using multi-tier assignment (this is the same as the Multi-Tier flow but it operates within the Community).
- Automation of the registration process. The flows will automatically register the assets if the following conditions happen:
  a. When all the tabs approved
  b. When the last tier in a Multi-tier process is completed
  c. Or whichever happens first.

The Communities are configured within the flow configuration and Asset Types, Producing Projects, etc., can point to a Community.

The following flowchart demonstrates how a Community for an asset is located by the flow, as well as how the rules for automatic acceptance are located by the flow.

Figure 5-2 Automatic Asset Acceptance Flowchart



Note: The same flowchart applies for automatic Registration. Simply substitute `autoRegister` for `autoAccept`.

Setting the Community for an ALER Project

Define the community for a project using the <producingProjectSettings> element. The following example demonstrates creating a project named “Registry” for the “SOA Center of Excellence” community, and with an ID of “40000”.

```
<producingProjectSettings>
   <producingProject name=”Registry” community=”SOA Center of Excellence
   id=”40000”/>
</producingProjectSettings>
```

Setting the Community for an Asset Type

Define the community for an Asset Type using the <assetType> element. The following example demonstrates creating an asset type named “Application” for the “SOA Center of Excellence” community, and with an ID of “158”.

```
<assetType name=”Application” community=”SOA Center of Excellence
id=”158”>
  <allTabs>
```

Configuring a Community to Automatically Accept an Asset

The following example demonstrates how to set the “SOA Center of Excellence” community to automatically accept assets.

```
<communities name=”SOA Center of Excellence autoAccept=”true”>
```

Note: Do not enable the “Community Acceptance” or the “Automated Acceptance” flows if repository users submit assets via the “Submit an Asset” link. This configuration is not currently supported in ALER.

Configuring a Community to Assign Assets for Tab Approval

If the AssetSubmitted event is wired to the Community flow, then the <approvers> element lists the approvers that will be assigned by the Community flow automatically.

```
<communities name=”Java” autoAccept=”true”>
  <approvers>
    <alerid>5003</alerid>
    <alerid>5004</alerid>
  </approvers>
```

For instructions on using the <alerid> in Tab Approval flows, see "Using an <alerid> for Tab Approvals."

Configuring a Community to Assign Assets for Tab Approval Using Multi-tier

Multi-tier assignment is the same as the Multi-Tier flow but it operates within the Community. For more information on the Multi-tier flow, see "Multi-tier Automatic Assignment Flows" on page 5-13.

Note: The tabs that are provided within the Multi-tier configuration of a community should be the common tabs that exist in all the asset types.

Configuring a Community to Automatically Register an Asset

The following example demonstrates how to set the "SOA Center of Excellence" community to automatically accept and register assets.

```
<communities name="SOA Center of Excellence autoAccept="true"
 autoRegister="true">
```

Configuring a Community to Have a Dedicated Registrar

The Registrar user name and password is required to accept, assign, and register assets. The Community flow will load the registrar information from the Community that the asset belongs to. If an asset does not belong to a community or if the registrar information is not found in the community, then the global registrar will be used by the Community flow.

The following is the order of precedence in getting the Community tag by the Community flows, as illustrated in Figure 5-1:

- Community Tag in the incoming event
- Community Tag in the Asset Type that the incoming asset belongs to
- Community Tag in the Producing Project that the incoming asset belongs to

Configuring Automated Acceptance and Automated Registration Flows

Besides using the Community flows to automatically accept and register assets, the following rules can be used to accept and register assets, as illustrated in Figure 5-1.

Note: Do not enable the “Community Acceptance” or the “Automated Acceptance” flows if repository users submit assets via the “Submit an Asset” link. This configuration is not currently supported in ALER.

Asset Type

The `autoAccept` and `autoRegister` flag within the `AssetType` element can be used to automatically accept or register assets.

```
<assetType name=”Application” autoAccept=”true” autoRegister=”true”
id=”158”>
  <allTabs>
    <tab name=”Oveview”/>
    <tab name=”Application Lifecycle”/>
  </allTabs>
```

Categorization Settings

By default the flows do not look for the `autoAccept` and `autoRegister` flags, since the look-up may affect performance. However, this can be enabled by using the `<action>` flag.

As shown in this example, the `<action>` flag must be set to `true` if the flows should use the Categorization settings. If not, the Categorization settings will be ignored.

```
<catgorizationTypeSettings action=”true”>
  <catgorizationType name=”AssetFunction” type “100”>
    <catgorizations name=”Application Adapters” autoAccept=”false”/>
    <catgorizations name=”Customer Information Acquisition”
autoAccept=”false”/>
    <catgorizations name=”eCommerce Frameworks” autoAccept=”false”/>
  </catgorizationType>
```

Submitter Role

The submitter role can be used to automatically accept or register the asset. If the role specified in the following configuration matches the submitter role, then the asset will be automatically accepted.

```
<automation>
  <autoRoles>
    <role>admin</role>
    <role>accesAdminstrator</role>
```

```
    </autoRoles>
    <autoApprovalTabs>
      <tab name="Documentation"/>
    </autoApprovalTabs>
  </automation>
```

Conflict Resolution and Precedence

In some cases, there will be more than one rule that matches for a given event trigger, so there is a hierarchy for how each rule is evaluated by the Automated Acceptance and Automated Registration flows for acceptance, registration, etc., as illustrated in Figure 5-1. The flow will scan for the following piece of metadata and as soon as it encounters the one in the following precedence, it will break and use the settings in that metadata.

- AssetType settings in the Flow configuration file
- Community Tag found in the incoming asset
- Community Tag found in the AssetType settings in the Flow configuration file
- Community Tag found in the ProducingProject settings in the Flow configuration file
- Categorization settings in the Flow configuration file
- SubmitterRole settings in the Flow configuration file

Multi-tier Automatic Assignment Flows

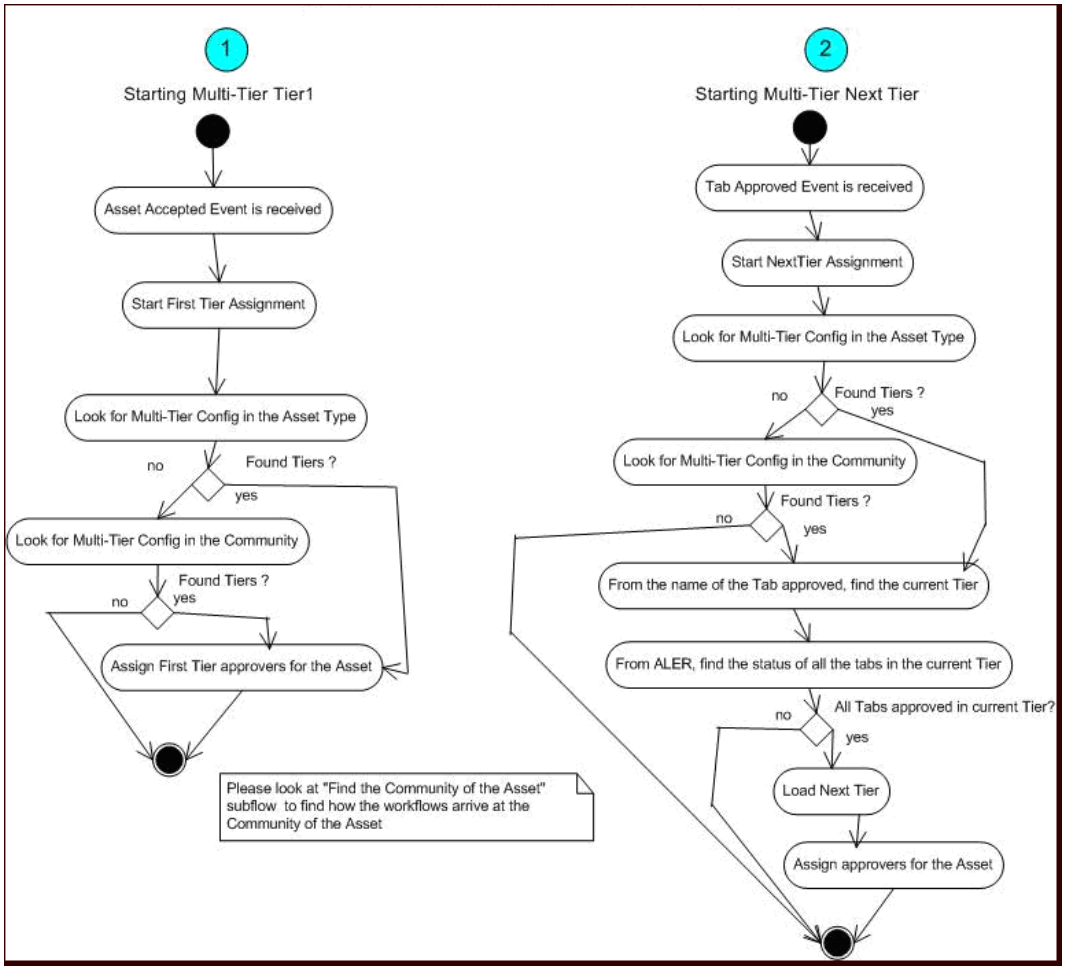
Multi-tier flows structure the asset tab approval process in multiple steps called *tiers*. Asset approval tabs can be grouped in tiers, and the Mult-tier flow tracks each tier to verify whether all the tabs are approved by the designated approvers. As soon as the last tab in a tier is approved, the Mult-tier flow starts the next tier by assigning the asset to the next level of designated approvers.

Use Cases

- In some cases, it may be desired to assign tabs for Tab Approval in multiple steps called *Tiers*. For example, it may be desirable to approve the Architecture tab first before approving the Documentation tab. This is because any architectural issue needs to be corrected first before it comes to the attention of the Documentation expert.
- In previous releases, Tab Approval was done manually by the registrar by manually tracking the status of each tab approval and then assigning the tabs for the next tier level approvals. With the Multi-tier flows, this process is automated by the flows.

The following flowchart demonstrates the flow of the Mult-tier process.

Figure 5-3 Multi-tier Automatic Assignment Flowchart



Using an <alerid> for Tab Approvals

When the `workflow.xml` file is generated, the following XML section is created under the `<allAssetSettings>` section. These are all the users that are created in ALER.

```
<alerUsers>
  <user name="admin" alerid="99"/>
  <user name="allpriv" alerid="50000"/>
  <user name="nopriv" alerid="50001"/>
  <user name="tier1" alerid="50002"/>
  <user name="tier2" alerid="50003"/>
  <user name="mrsmith" alerid="50004"/>
</alerUsers>
```

As the Workflow Administrator, you need to identify the user(s) by name that you want to use for approving the asset tabs and use the corresponding `<alerid>`. Then you can use that `<alerid>` in the Workflow XML, such as in the following Multi-tier approval flow:

```
<tiers>
  <tier name="Tier1">
    <approvers>
      <alerid>50001</alerid>
    </approvers>
    <tabs>
      <tab name="Overview"/>
      <tab name="Technical"/>
      <tab name="Documentation"/>
    </tabs>
</tier>
```

Setting Up a Community for Multi-tier Tab Approval

The following example demonstrates how the Multi-tier flow is configured for tab approvers in the “SOA Center of Excellence” community to automatically accept tabs.

```
<communities name=”SOA Center of Excellence autoAccept=”true”>
  <tiers>
    <tier name=”Tier1”>
      <approvers>
        <alerid>5002</alerid>
      </approvers>
      <tabs>
        <tab name=”Overview”>
        <tab name=”Taxonomy”>
      </tabs>
```

```
      </tier>
      <tier name=”Tier2”>
        <approvers>
          <alerid>5003</alerid>
        </approvers>
        <tabs>
          <tab name=”Architecture”>
        </tabs>
      </tier>
    </tiers>
  </communities>
```

Note: Tabs that are provided within the Multi-tier configuration of a Community should be the common tabs that exist in all the Asset Types.

Setting Up an Asset Type for Multi-tier Tab Approval

The following example demonstrates how the tabs of an asset type of “Application” are configured for multi-tier approval.

```
<assetType name=”Application” id=”158”>
  <allTabs>
    <tab name=”Oveview”/>
    <tab name=”Application Lifecycle”/>
    <tab name=”License Information”/>
    <tab name=”Certification Tracking”/>
    <tab name=”Taxonomy”/>
    <tab name=”Documentation”/>
    <tab name=”Relationships”/>
    <tab name=”Support”/>
    <tab name=”Cost Categories”/>
    <tab name=”Ownership”/>
    <tab name=”Technology Stack”/>
    <tab name=”Operational Information”/>
    <tab name=”Miscellaneous”/>
  </allTabs>
  <tiers>
    <!--Please change “_CHANGE_TIER1_NAME_” to the name of the Tier-->
    <!--Example:- “Tier1”-->
```

```
  <tier name="Tier1">
    <approvers>
      <alerid>99</alerid>
    </approvers>
    <tabs>
    <!--Please change "_CHANGE_TABNAME_" to the name of the Tab-->
    <!--Example:- "Documentation"-->
      <tab name="Overview">
      <tab name="Taxonomy">
    </tabs>
  </tier>
</tiers>
```

Metadata Change Flows

Metadata flows are a group of flows that take one or more actions when a metadata element of an asset changes. The Metadata element that changes will trigger an event that is wired to one or more flows. For instructions on how to wire an event to a flow, see "Wiring Asset Events to Flows."

Use Cases

These are some of the use cases where Metadata Change Flows may apply:

- When the "Asset Lifecycle Stage" metadata element of an asset changes from "Build" to "Release,", you may want to change Custom Access Settings to have more restricted access control to the asset.
- When the "Name" of an asset changes, you may want to notify the subscribers.
- When any metadata element of an element changes, you may want the asset to go through a "Change Management" approval process. The "Change Management" will involve the following:
  - Unapprove a tab named "Change Management"
  - Assign the asset to the registrar
  - Append the kind of change to a field called "Reason for reassignment" to assist the registrar

Configuring Metadata Change Flows

Available Metadata Change Events/States

Following are the states that are available that can be wired to Metadata Change flows.

Note: Besides these events, any categorization changes can be wired, including the custom categorization.

```
  <state name="urn:com:bea:aler:events:type:MetaDataChange:name">
  <state name="urn:com:bea:aler:events:type:MetaDataChange:version">
  <state name="urn:com:bea:aler:events:type:MetaDataChange:description">
  <state name="urn:com:bea:aler:events:type:CategorizationChanged:
  assetLifecycleStage"/>
  <state
name="urn:com:bea:aler:events:type:CategorizationChanged:classification">
  <state name="urn:com:bea:aler:events:type:MetaDataChange:supported">
```

```
  <state
name="urn:com:bea:aler:events:type:MetaDataChange:organizationalOwnership">
  <state name="urn:com:bea:aler:events:type:MetaDataChange:usageFee">
```

Available Flows That Can Be Wired to Actions

These are the pre-defined flows that can be wired to actions. These flow names should appear as content inside the `<action>` element to indicate that this is the action that should take place when the event occurs. Note that any element other than `<action>` are parameters used by specific flows.

- `ChangeCAS` – applies one or more Custom Access settings to an asset
- `ChangeAssetLifecycle` – sets the Asset Lifecycle Stage of an asset
- `ChangeClassification` – sets the classification of an asset
- `ReAssignAssetToRegistrar` – assigns the asset to Registrar
- `AddCommunityTag` – saves the "Community" of an asset to ALER
- `NotifySubscriber` – notifies the Subscribers about the Metadata Change
- `NotifyRegistrationActors` – notifies the Registrar, Subscribers, Owners, etc., about the Metadata Change
- `NotifyCustomUser` – notifies configured custom users about the Metadata Change
- `UnapproveChangeManagementTab` – triggers the Change Management process
- `ResetChangeManagementTab` – resets the "Reason for reassignment" field in the Change Management tab as soon as the Change Management tab is approved
- `CommunityAccept` – invokes the Community Accept Flow used when an asset is submitted
- `CommunityAssign` – invokes the Community Assign Flow used when an asset is accepted
- `MultiTier_Tier1_Assign` – invokes the Multi-Tier Flow used when an asset is accepted
- `MultiTier_NextTier_Assign` – invokes the Multi-Tier Flow used when a tab is approved
- `ApproveTabAction` – approves one or more tab
- `UnapproveTabAction` – unapproves one or more tab

- `AutoApproveTabAction` – approves one or more configured tab based on the role of the submitter
- `AllTabsApproval_Register` – invokes the flow to register the asset when all the tabs are approved
- `ReAssignAssetToRegistrar` – Assigns the asset to the Registrar for approval. The flow uses the Community Registrar if one is configured. If not, it uses the Global Registrar.
- `ResetFlowState` – Resets the State information used by the Timer based flows. This is useful in cases where a Timer flow is tracking the Unsubmitted assets and when the state changes from Unsubmitted to submitted, so the State information can be reset. If not reset, then if the asset goes back to Unsubmitted, the workflows use the same state that was previously set. This is not always desirable and the ResetFlowState action can be used in appropriate events or states to reset the state information.
- `UnRegisterAssetAction` – Unregisters the Asset if the asset is in registered state.

Example Metadata Change Configuration

This sample configuration specifies that when an asset is registered, it invokes two flows by the names of "NotifySubscriber" and "ChangeCAS." Note that the element `<customAccessSettings>` is a parameter to the flow `ChangeCAS`, which tells the flows the names of the CAS that should be applied.

```
<state name="urn:com:bea:aler:events:type:AssetRegister">
  <action>NotifySubscriber</action>
  <action>ChangeCAS</action>
  <customAccessSettings>
    <customAccessSetting>MyCAS</customAccessSetting>
    <customAccessSetting>MyCAS2</customAccessSetting>
  </customAccessSettings>
</state>
<state name="urn:com:bea:aler:events:type:AssetUnAccept">
  <action>NotifySubscriber</action>
  <action>ChangeClassification</action>
  <classification>Approved</classification>
</state>
```

Example Metadata Change Configuration That Checks for Metadata Value

It is also possible to invoke a flow not only when a metadata element changes, but also when it takes a specific value. For example, when the "Asset Lifecycle Stage" metadata element of an asset changes from "Build" to "Release," you may want to apply one set of Custom Access Settings, where as when the value changes from "Plan" to "Build," you may want to apply a different set. Here is an example:

```
  <state
name="urn:com:bea:aler:events:type:CategorizationChanged:AssetLifecycleSta
ge" value="Stage 4 - Release">
    <action>ChangeCAS</action>
    <customAccessSettings>
      <customAccessSetting>MyCAS</customAccessSetting>
    </customAccessSettings>
  </state>
  <state
name="urn:com:bea:aler:events:type:CategorizationChanged:AssetLifecycleSta
ge" value="Stage 3 - Build">
    <action>ChangeCAS</action>
    <customAccessSettings>
      <customAccessSetting>MyCAS2</customAccessSetting>
    </customAccessSettings>
  </state>
```

ChangeClassification

Sets the classification of an asset. ChangeClassification uses the following element to set the classification.

```
  <state name="urn:com:bea:aler:events:type:AssetRegister">
    <action>ChangeClassification</action>
    <classification>Approved</classification>
  </state>
```

ChangeCAS

Applies one or more Custom Access Settings to an asset. ChangeCAS uses the following element to set the custom access settings.

```
<state name=”urn:com:bea:aler:events:type:AssetRegister”>
  <action>ChangeCAS</action>
  <customAccessSettings>
    <customAccessSetting>MyCAS</customAccessSetting>
    <customAccessSetting>MyCAS2</customAccessSetting>
  </customAccessSettings>
</state>
```

ChangeAssetLifecycle

Sets the Asset Lifecycle stage of an asset. ChangeAssetLifeCycle uses the following element to set the asset life cycle.

```
<state name=”urn:com:bea:aler:events:type:AssetRegister”>
  <action>ChangeAssetLifeCycle</action>
  <assetLifeCycle>Stage 3 - Build</assetLifeCycle>
</state>
```

ApproveTabAction

The ApproveTabAction flow approves one or more tabs of an asset. The following configuration, approves the “Overview” and “Taxonomy” tabs.

```
<state name=?urn:com:bea:aler:events:type:MetaDataChange:name?>
  <action>ApproveTabAction</action>
  <approveTabs>
    <tab name=?Overview?>
    <tab name=?Taxonomy?>
 </approveTabs>
</state>
```

UnapproveTabAction

The following element configures the list of tabs to be unapproved by the `UnapproveTabAction` flow.

```
<state name=”urn:com:bea:aler:events:type:MetaDataChange:name”>
  <action>UnApproveTabAction</action>
  <unapproveTabs>
    <Tab name=”Overview”>
    <Tab name=”Taxonomy”>
```

```
  </unapproveTabs>
</state>
```

AutoApproveTabAction

The AutoApproveTabAction flow approves tabs based on the role of the submitter. For example, the following element under `<allAssetSettings>` configures the list of tabs that need to be automatically approved based on the role of the submitter. The roles that are acceptable are also configured.

```
<automation>
  <autoRoles>
    <role>admin</role>
    <role>accesAdminstrator</role>
  </autoRoles>
  <autoApprovalTabs>
    <tab name=”Documentation”/>
  </autoApprovalTabs>
</automation>
```

Here is the configuration for invoking the flow:

```
<state name=”urn:com:bea:aler:events:type:AssetRegister”>
  <action>AutoApproveTabAction</action>
</state>
```

UnapproveChangeManagementTab

When any metadata element of an element changes, you may want the asset to go through a “Change Management” approval process, which involves following.

- Unapprove a tab by name “Change Management”
- Assign the asset to the registrar.
- Append the kind of change to a field called “Reason for reassignment” to assist the registrar

```
<state name=”urn:com:bea:aler:events:type:MetaDataChange:name”>
  <action>UnApproveChangeManagementTab</action>
</state>
```

ResetChangeManagementTab

This flow resets the “Reason for reassignment” field in the Change Management tab as soon as the Change Management tab is approved.

```
<state name=”urn:com:bea:aler:events:type:AssetTabApproved”>
  <action>MultiTier_NextTier_Assign</action>
  <action>ResetChangeManagementTab</action>
</state>
```

NotifyCustomUser

Notifies configured custom users about the metadata change. The email addresses of the users are configured inside the `<customNotification>` element under `<allAssetSettings>`, as shown below:

```
<allAssetSettings>
  <notification timerInterval=”id”>
    <customNotification>
      <emailAddress>smith@bea.com</emailAddress>
    </customNotification>
  </notification>
```

Invoking Flows Based on Approval of Named Tabs

A metadata change flow can be executed based on the approval of a specific tabs, as follows:

```
  <state name="urn:com:bea:aler:events:type:AssetTabApproved" value
="Overview">
    <action>MultiTier_NextTier_Assign</action>
    <action>ChangeAssetLifecycle</action>
    <assetLifecycle>Stage 3 - Build</assetLifecycle>
  </state>
```

Time-based Escalation Flows

The Time-based Escalation flows track assets in various states and notifies all interested parties. The following section explains how to configure the Time-based Escalation flows. There are four different kinds of Time-based Escalation flows and each one can be configured individually, as described in the following sections.

Open the `workflow.xml` configuration file and locate the `<notification>` element.

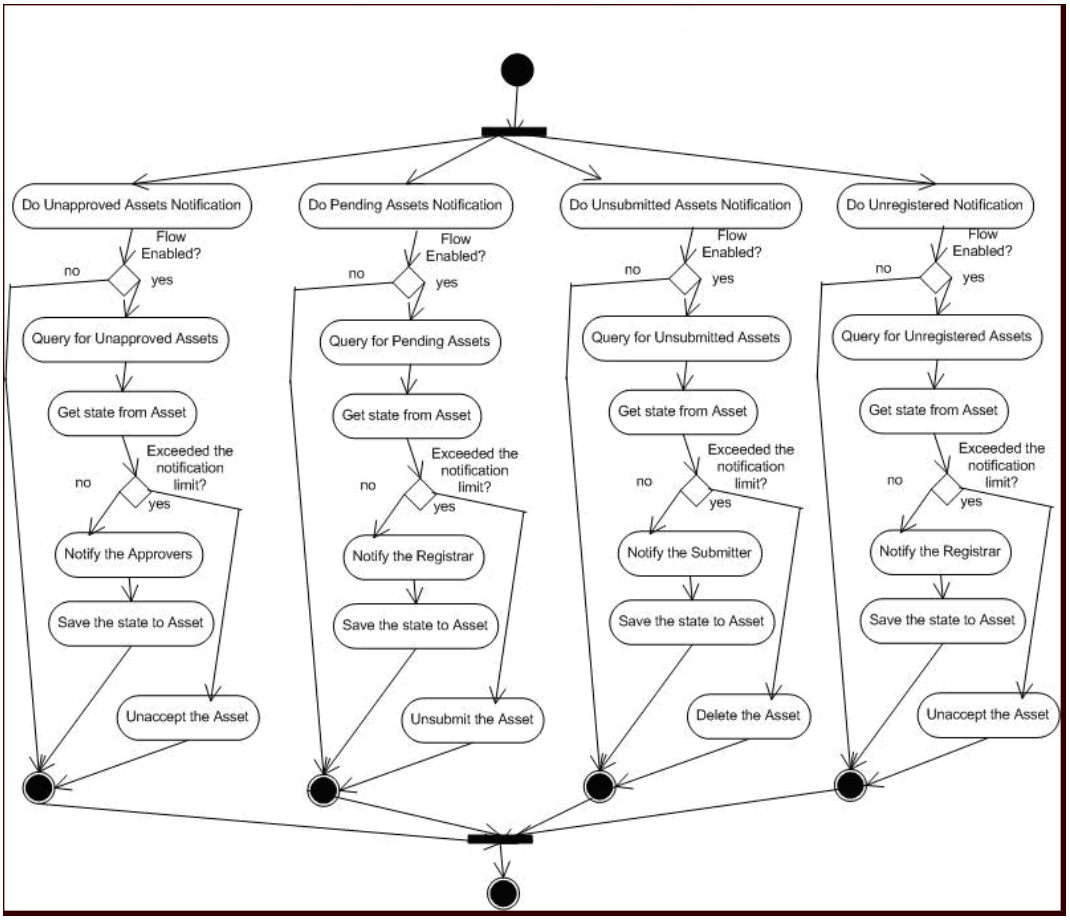
```
<notification timerInterval=”1d”>
  <numTimesNotify>10</numTimesNotify>
  <daysBeforeNextNotification>2</daysBeforeNextNotification>
```

- The **timerInterval** element specifies the time interval after which the flows will be triggered. In a production environment, this *should* be set to `"1d"`, which means the flows will be triggered once a day. However for testing purposes, you can set it to `"1m"` or `"5m"` to trigger the flows every minute or every five minutes. Also, each time this field is changed, the Event Engine needs to be restarted, unlike the other field changes that can be refreshed using the refresh tool.
- The **numTimesNotify** element specifies how many times the notifications should be sent by the Time-based Escalation flows.
- The **daysBeforeNextNotification** element specifies how many days need to elapse in between the notifications.

  Note: If the **timerInterval** element is configured in minutes to trigger flows in minute intervals for testing purposes, then the specified interval for **daysBeforeNextNotification** will also be interpreted in minutes.

The following flowchart demonstrates the flow of the Time-based Escalation flows.

Figure 5-4 Time-based Escalation Flowchart



Tracking Unsubmitted Assets

This flow tracks assets that are in an “unsubmitted” status and sends notification to the owners to take action.

```
  <owner_resubmit action=”false” days=”0” regressOnInaction=”true”
queryOperator=”eq”/>
```

- `action="true"` enables the flow and `action="false"` disables the flow.
- `days="10"` tracks the assets that reached unsubmitted status 10 days ago. The Time-based Escalation flows use the current date and subtracts the value from this attribute.
- `regressOnInaction="true"` regresses the asset on inaction. For example, unsubmitted assets may be deleted.
- `queryOperator="eq"` uses the equals operator when the date is used for querying. Other possible values are `"lte"`, `"gte"`, etc.

Tracking Unaccepted Assets

This flow tracks assets that are in an “unaccepted” status and sends notification to the registrar to take action.

```
  <registrar_accept action=”false” days=”0” regressOnInaction=”true”
queryOperator=”eq”/>
```

- `action="true"` enables the flow and `action="false"` disables the flow.
- `days="10"` tracks the assets that reached unsubmitted status 10 days ago. The Time-based Escalation flow use the current date and subtracts the value from this attribute.
- `regressOnInaction="true"` regresses the asset on inaction. For example, submitted assets may be unsubmitted.
- `queryOperator="eq"` uses the equals operator when the date is used for querying. Other possible values are `"lte"`, `"gte"`, etc.

Tracking Unapproved Assets

This flow tracks assets that are in an “unapproved” status and sends notification to the approvers to take action.

```
  <assignees_approve action=”false” days=”0” regressOnInaction=”true”
queryOperator=”eq”/>
```

- `action="true"` enables the flow and `action="false"` disables the flow.
- `days="10"` tracks the assets that reached unsubmitted status 10 days ago. The Time-based Escalation flow use the current date and subtracts the value from this attribute.
- `regressOnInaction="true"` regresses the asset on inaction. For example, accepted assets may be unaccepted.

- `queryOperator="eq"` uses the equals operator when the date is used for querying. Other possible values are `"lte"`, `"gte"`, etc.

Tracking Unregistered Assets

This flow tracks the assets that are in an “unregistered” status and sends notification to the approvers to take action.

```
  <registrar_register action=”false” days=”0” regressOnInaction=”true”
queryOperator=”eq”/>
```

- `action="true"` enables the flow and `action="false"` disables the flow.
- `days="10"` tracks the assets that reached unsubmitted status 10 days ago. The Time-based Escalation flow use the current date and subtracts the value from this attribute.
- `regressOnInaction="true"` regresses the asset on inaction. For example, accepted assets may be unaccepted.
- `queryOperator="eq"` uses the equals operator when the date is used for querying. Other possible values are `"lte"`, `"gte"`, etc.

Validation Expiration Flows

The Validation Expiration flows track the expired assets prior to the expiration date, as well as on the date of expiration, and sends warning notifications to all interested parties. After *X* number of days of expiration, the flows unregister the assets. After *Y* number of days of expiration, the flows deactivate the assets. After *Z* number of days of expiration, the flows delete the assets.

```
<notification timerInterval=”1d”>
  <numTimesNotify>10</numTimesNotify>
  <daysBeforeNextNotification>2</daysBeforeNextNotification>
```

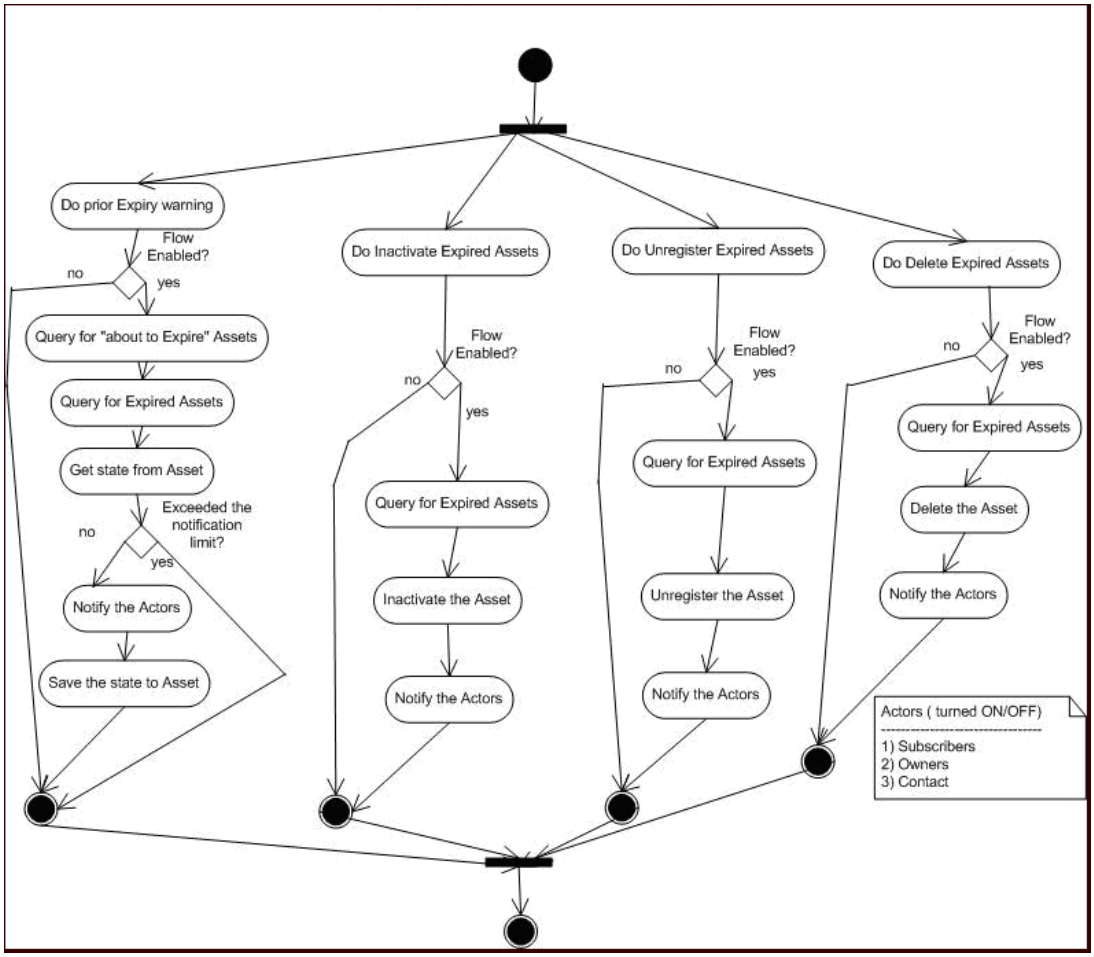
- The `timerInterval` attribute configure the time interval that the flows will be triggered. This *should* be set to `"1d"`, which means the interval is one day. However for testing, this can be set to `"1m"` or `"5m"` to trigger every minute or every 5 minutes. Also, every time this field is changed, the Event Engine needs to be restarted, unlike the other field changes that can be refreshed using the refresh tool.
- The `numTimesNotify` element specifies how many times the notifications should be sent by the Validation Expiration flow.

- The `daysBeforeNextNotification` element specifies how many days need to elapse in between the notifications.

```
  <expiration>
    <expiration_warning action=”false” days=”10” owner=”false”
subscriber=”false” contact=”99”/>
    <unregister_after_expire action=”true” days=”10” queryOperator=”eq”/>
    <inactive_after_expire action=”true” days=”10” queryOperator=”eq”/>
    <delete_after_expire action=”true” days=”10” queryOperator=”eq”/>
  </expiration>
```

The following flowchart demonstrates the flow of the Time-based Escalation flows.

Figure 5-5 Validation Expiration Flowchart



Asset Expiration Warning Notification

The following line enables the warning notification and determines who should receive the notifications.

```
  <expiration_warning action=”false” days=”10” owner=”false”
subscriber=”false” contact=”99”/
```

Note: The `days` element configures the number of days prior to the expiration that the warning should be sent.

Unregister Assets After Expiration

The following line enables the Metadata Change flow to unregister the asset after 10 days of expiration.

```
<unregister_after_expire action=”true” days=”10” queryOperator=”eq”/>
```

Inactivate After Expiration

The following line enables the Metadata Change flow to inactivate the asset after 10 days of expiration

```
<inactive_after_expire action=”true” days=”10” queryOperator=”eq”/>
```

Delete Assets After Expiration

The following line enables the Metadata Change flow to delete the asset after 10 days of expiration:

```
<delete_after_expire action=”true” days=”10” queryOperator=”eq”/>
```

Customizing Flow Notification Email Templates

The Automated Registration Flows automatically send email notifications under many circumstances. There are five new email templates for the new flows. The email templates are stored within ALER and the flows invoke an ALER API by passing name/value pairs that are then substituted by ALER.

Administrators can customize the email subject, body, etc., the same way as other email templates. The following are the templates that are used by the Advanced Registration Flows:

- Metadata of asset has changed – Notifies the registrar and the users assigned to the asset that the metadata has changed.
- Registration status unchanged – Notifies the registrar and the users assigned to the asset that the registration status `<%asset.reg.status%>` has remained unchanged for more than `<%action.pending.days%>` days.
- Status of expired asset has changed – Notifies the registrar and the users assigned to the expired asset that the status has changed.
- Prior to expiration – Notifies the registrar and the users assigned to the asset that it is due for expiration.
- Asset has been expired – Notifies the registrar and the users assigned to the asset that it has been expired.

For more information about email templates, refer to the *ALER Administration Guide*.

CHAPTER 6

Configuring JMS Servers for ALER

This section contains information on the following subjects:

- Overview of JMS for the Event Manager
- Configuring Connectivity Properties for External JMS Servers
- Configuring the Embedded ActiveMQ JMS Server to Use a Database
- Configuring JMS Durable Subscribers for Web Service Endpoints
- Configuring JMS Servers In an ALER Cluster
- Configuring a JMS Provider In WebSphere 6.1.0.5

Overview of JMS for the Event Manager

The Event Manager uses an embedded version of Apache ActiveMQ JMS Server that is enabled by default. The embedded JMS Server is configured to run out-of-the-box without any additional configuration. However, if an external JMS server is preferred, such as BEA Weblogic Server JMS or IBM WebSphere Application Server, then a number of ALER system settings must be configured.

Note: When ALER is deployed on WebSphere 6.x, the embedded Apache ActiveMQ JMS Server cannot be used due to conflicts in the classes used by ActiveMQ and ALER. Therefore, WebSphere 6.x customers should use the default JMS implementation that comes with WebSphere 6.x. See Configuring a JMS Provider In WebSphere 6.1.0.5.

Configuring Connectivity Properties for External JMS Servers

ALER's System Settings section allows administrators to configure the basic ALER operation and to enable/disable specific features. The Event Manager's JMS-related settings are under the "Eventing" group under the main "External Integrations" category. For more information about System Settings, see the *ALER Administration Guide*. Additional "Eventing" properties are described in Configuring the ALER Event Manager.

Enabling and Configuring an External JMS Server

The internal Apache ActiveMQ JMS Server needs to be disabled in order to configure an external JMS product. You must also configure JNDI and JMS properties for the external JMS.

Note: These steps are for configuring a single external JMS server. For instructions on configuring multiple JMS servers in a cluster, see Configuring JMS Servers In an ALER Cluster.

1. Click **System Settings** in the sidebar on the ALER **Admin** screen.
2. Enter **Event** in the System Settings Search box to view all the Event Manager related settings.
3. Disable the internal JMS server by clicking **False** next to the **Event Manager Embedded JMS Enable** property. This forces the Event Manager to use an external JMS server.
4. Configure the required JNDI properties:
   - **JNDI URL** – Specifies the JNDI URL. For example, `t3://localhost:7001`.

- **JNDI User Name** – Specifies the JNDI user name.
- **JNDI Password** – Specifies the password for the JNDI User Name.
- **JNDI Context Factory** – Specifies the JNDI initial context factory. For example, `weblogic.jndi.WLInitialContextFactory`.

5. Configure the following JMS properties:
   - **JMS Connection Factory** – Specifies the JMS connection factory to enable JMS clients to create JMS connections. For example, `weblogic.examples.jms.TopicConnectionFactory`.
   - **JMS Topic** – Specifies the JMS topic, which is a publish/subscribe destination type for a JMS server. For example, `weblogic.examples.jms.TopicConnectionFactory`.
6. Click **Save**.
7. Restart ALER for the configuration changes to take effect.

Configuring JMS Message Header Properties

Every JMS message contains a standard set of header fields that is included by default and available to message consumers. The Message Expiration and Delivery Mode headers can be configured using the ALER System Settings.

1. Access the "Eventing" System Settings, as described in "Enabling and Configuring an External JMS Server" on page 6-2.
2. Configure the JMS message header properties:
   - **JMS Message Expiration** – Sets the JMS message expiration time in seconds. If set, unprocessed events will expire in the specified number of seconds. The default is 0 seconds, which means that messages will never expire. However, some environments have policies that require that JMS messages cannot be stored forever if they are not selected for some reason.
   - **JMS Delivery Mode** – Sets the JMS message delivery mode to either PERSISTENT or NON-PERSISTENT values. If set to PERSISTENT, the JMS server will write the events to the underlying store. Although more reliable, persisting events to a store can affect performance. The default is PERSISTENT.
3. Click **Save**.
4. Restart ALER for the configuration changes to take effect.

Miscellaneous JMS Properties

Note: You must restart ALER after changing any Eventing property in order for the changes to take effect.

The following miscellaneous System Settings can also be configured.

- **Event Manager JMS Subscribers Enabled** – If set to **False**, then the internal JMS subscribers will not be enabled. This is to make sure that the embedded JMS server is started, but an external tool can be used to connect to the embedded server using the given durable subscriber name and the stored events can be cleaned up.
- **JMS Subscribers Client ID** – Specifies the JMS durable subscriber ID. For example, `ALER_JmsSubscriber`.
- **JMS Producers Client ID** – Specifies the JMS producer's client ID. For example, `ALER_DeliveryManager`.
- **Lazy Initialize Event Engine** – When enabled, the Event Manager will be initialized when an event is produced for the first time. This property should be enabled for either of the following reasons:
  - If there is a large number of events stored by the JMS server *and* if it is required that these events should not be processed as soon as ALER is started.
  - There are startup issues that occur because of the timing of initializing the embedded JMS server.

Configuring External JMS Jar Files

If an external JMS server is being used, then the external JMS server-related `JAR` files should be copied to the `WEB-INF\lib` directory.

Configuring the Embedded ActiveMQ JMS Server to Use a Database

By default, the ActiveMQ JMS server uses a file-based store to store events. However, you can specify to have events stored in a database. Simply, configure the `activemq.xml` file in the `WEB-INF\classes` directory to use your database parameters.

For example:

```
 <persistenceAdapter>
   <journaledJDBC journalLogFiles="5" dataDirectory="../activemq-data"
dataSource="#oracle-ds" />
   <!-- To use a different datasource, use the following syntax : -->
   <!-- <journaledJDBC journalLogFiles="5" dataDirectory="../activemq-data"
dataSource="#postgres-ds"/> -->

 <!--  Oracle DataSource Sample Setup  -->
- <bean id="oracle-ds" class="org.apache.commons.dbcp.BasicDataSource"
destroy-method="close">
  <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver" />
  <property name="url" value="jdbc:oracle:thin:@localhost:1521:AMQDB" />
  <property name="username" value="scott" />
  <property name="password" value="tiger" />
  <property name="poolPreparedStatements" value="true" />
  </bean>
```

Configuring JMS Durable Subscribers for Web Service Endpoints

The Event Manager creates one durable subscriber for each Web Service endpoint it encounters in the Subscription Manager XML file. This ensures that events are stored if the endpoints are not online and that they can be reliably delivered once the endpoints are online again. As per the JMS Specification, the durable subscriber name should be unique across the JMS server. The Event Manager gets the durable subscriber name from the `name` field found in the `EndPointEventSubscription.xml` file, as shown in this example:

```
<sub:EventSubscriptionData
xmlns:sub="http://www.bea.com/infra/events/subscription"
  <sub:eventSubscription>
    <sub:endPoint name="ALBPMEndpoint">
```

Note: JMS servers associate the durable subscriber name with the message selectors. Therefore, if the message selector is changed, either a new durable subscriber name should be provided or the existing one should be deleted. You can use the ALER "Event Cleanup" tool, as described in "Cleaning Up Stored Events." You can also use a JMS-specific tool to accomplish this.

Configuring JMS Servers In an ALER Cluster

Note: Before you begin, refer to the *ALER Clustering Guide* for information on configuring ALER in a clustered environment.

Enabling JMS Clustering Mode

If ALER is deployed on cluster mode, you must enable clustering on each ALER instance regardless of which type of JMS server being used (embedded or external).

1. Click **System Settings** in the sidebar on the ALER **Admin** screen.
2. Enter **cmee.eventframework.clustering.enabled** in the Enable New System Setting box and click **Enable** to reveal this hidden property.
3. Set the **Clustering Enabled** property to **True**.
4. Set other required properties based on the type of JMS server, as described in the following sections.

Configuring Embedded JMS Servers for Clustering

In a clustered environment, each member ALER instance in the cluster will have one embedded JMS server. For example, in case of two-node cluster, there will be two ALER instances, such as `server01` and `server02`, with each having one embedded JMS server. Once clustering is enabled for the embedded JMS servers, you then need to specify the connection URL information for the embedded JMS servers on `server01` and `server02`.

1. Click **System Settings** in the sidebar on the ALER **Admin** screen.
2. Enter **cmee.eventframework.embedded.jms.url** in the Enable New System Setting box and click **Enable** to reveal this hidden property.
3. In the **Embedded JMS Server URL** property, supply the connection URL for the embedded JMS servers on the clustered ALER servers, using the following format.

   `failover:(tcp://`
   `$SEVER_DNS_NAME_OR_IP$``:61700,tcp://`*`$SEVER_DNS_NAME_OR_IP$`*`:61700, …)`

   where:

   `$SEVER_DNS_NAME_OR_IP$` are replaced by actual server DNS name or IP address. The entries should be repeated for each ALER server in a given cluster.

Using the example above, this could be set to:
`failover:(tcp://server01:61700,tcp://server02:61700)`

Caution: Port 61700 is the default port for the embedded JMS server, and therefore should not be used by any other application on the ALER server unless another port is configured for the embedded JSM server.

4. Click **Save**.
5. Repeat steps 1-4 for each ALER instance in a given cluster. Using the example above, the Embedded Broker URLs could be set to:
`failover:(tcp://server01:61700,tcp://server02:61700)`

Tip: Make sure that each embedded JMS server is enabled by setting the **cmee.eventframework.embedded.jms.enabled** property to **True**.

Configuring External JMS Servers for Clustering

For external JMS servers, no additional configuration is required. However, you must make sure that the embedded JMS server is disabled, as follows:

1. Click **System Settings** in the sidebar on the ALER **Admin** screen.
2. Set the **Event Manager Embedded JMS Enable** property to **False** (i.e., **cmee.eventframework.embedded.jms.enabled** is **False**.

Configuring a JMS Provider In WebSphere 6.1.0.5

When ALER is deployed on WebSphere Application Server 6.1.0.5, the embedded Apache ActiveMQ JMS server cannot be used. Therefore, WebSphere 6.1.0.5 implementations must use the default JMS provider that comes with WebSphere 6.1.0.5.

To configure a JMS provider for ALER in WebSphere 6.1.0.5, complete the following steps in the WebSphere administration console and in your ALER application.

1. Create a new Service Integration Bus:
   a. In the navigation pane, expand **Service Integration**, and then click **Buses**.
   b. On the Buses page, click **New**.
   c. On the Create a new bus page, enter **alerbus** as the name for the new bus.
   d. Clear the **Bus security** check box.
   e. Click **Next**, and then click **Finish**.
2. Add a Bus member to the newly created *alerbus*:
   a. On the Buses page, click the **alerbus** link.
   b. Under the Topology category, click **Bus members**.
   c. On the Bus members page, click **Add**.
   d. On the Add a new bus member > Select Server, Cluster or WebSphere MQ server page, accept the default **Server** option and click **Next**.
   e. On the Add a new bus member > Select the type of message store page, accept the default **File store** option and click **Next**.
   f. On the Add a new bus member > Provide the message store properties page, accept the default values and click **Next**.
   g. On the Add a new bus member > Confirmation page, click **Finish**.
   h. On the Buses page, click **Save**.
3. Create a JMS Topic Connection Factory in the default message provider:
   a. In the navigation pane, expand **JMS**, and then click **JMS providers**.

b. Click the **Default messaging provider** option, with a Scope of `Node=<nodename>, server=server1`.

c. On the JMS providers > Default messaging provider page, click the **Topic connection factories** option under Additional Properties.

d. On the JMS providers > Default messaging provider > Topic connection factories page, click **New**.

e. On the Administration page, configure the topic connection factory as follows:

- Name – `alerEventingTopicCFDefault`
- JNDI name – `jms.alerEventingTopicCFDefault`
- Bus name – `alerbus`
- Client identifier – `ALER_JmsProducer`
- Durable subscription home – `<nodename>.server1-alerbus`

f. Click **Apply**, and then click **Save**.

4. Create a JMS Topic in the default message provider:

a. Re-navigate to the JMS providers > Default messaging provider page.

b. Click the **Topics** option under Additional Properties.

c. On the JMS providers > Default messaging provider > Topics page, click **New**.

d. On the Administration page, configure the topic as follows:

- Name – `alerEventingTopicDefault`
- JNDI name – `jms.alerEventingTopicDefault`
- Topic name – `alerEventingTopicDefault`
- Bus name – `alerbus`
- Topic space – `Default.Topic.Space`

e. Click **Apply**, and then click **Save** to save your changes.

5. Deploy the `aler.ear` application file, as follows:

a. In the navigation pane, expand **Applications**, and then click **Enterprise Applications**.

b. On the Enterprise Applications page, click **Install**.

c. On the Preparing for the application install page, click **Browse**, specify the `aler.ear` file in the path, and then click **Next**.

d. Click **Next** on the Select installation options page.

e. Click **Next** on the Map modules to servers page.

f. On the Map resources to resource references page, click **Browse** in the Target Resource JNDI Name column.

g. On the Enterprise application > Available resources page, select **alerEventingTopicCFDefault**, and then click **Apply**.

h. Click **Next** on the ensuing Map resources to resource references page.

i. On the Map resource environment entry references to resources page, enter `jms/aler/alerEventingTopicDefault` in Target Resource JNDI Name and then click **Next**.

j. Click **Finish** on the Summary page.

k. After the application is installed, click **Save** to save it to the Master Configuration.

6. Follow the "Manually Installing the ALBPM Process Engine and Advanced Registration Flows" steps in the ALER Installation Guide to deploy additional files in the `web-inf/classes` directory and the database drivers required by the ALER application.

7. Configure the ALER `eventing.properties` file for the WebSphere settings:

   a. Navigate to the `<ALER Domain>\WEB-INF\classes` directory.

   b. Use a text editor to modify the `eventing.properties` file as follows:

      - `cmee.eventframework.jms.topic=jms.alerEventingTopicDefault`
      - `cmee.eventframework.jndi.provider.url=iiop\://localhost:2809`
      - `cmee.eventframework.embedded.jms.enabled=false`
      - `cmee.eventframework.jndi.context.factory=com.ibm.websphere.naming.WsnInitialContextFactory`
      - `cmee.eventframework.jms.connection.factory=jms.alerEventingTopicCFDefault`

   c. Save the file.

8. Restart the WebSphere application server to enable the modified settings.

9. Check the WebSphere logs for possible errors:
   `\WebSphere\AppServer\profiles\AppSrv01\logs\server1`

CHAPTER 7

Monitoring and Managing Events

This section contains information on the following subjects:

- "Overview"
- "Monitoring Events"
- "Cleaning Up Stored Events"
- "Generating the Workflow Config File"
- "Refreshing the Workflow Config File"
- "Encrypting Your Passwords"

Overview

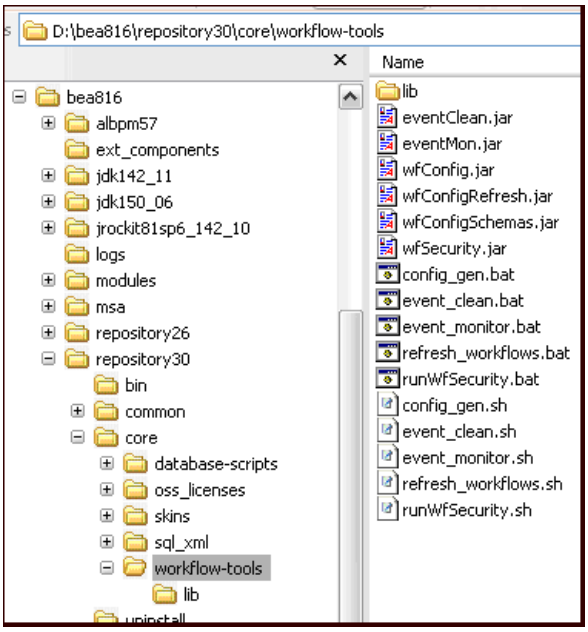
This document discusses how to use the administrative tools that are shipped as part of ALER3.0. The Advanced Registration Flow administrative tools are used to

- Monitor events using a command-line interface
- Clean up the events and unsubscribe the JMS durable subscriber
- Generate the Workflow Configuration file
- Refresh the ALBPM Engine with the latest Workflow Configuration file
- Encrypt the passwords stored in the Workflow Configuration and Subscription Manager files

The administrative tools are installed under the following directory:

<BEA Home>/repository30/core/workflow-tools

Figure 7-1 Location of Workflow Tools



Monitoring Events

The Event Manager has a tool for monitoring the events that are generated by the Event Manager. The tool peeks into the event traffic and prints information, such as the Event Body and Event Properties, as shown in this section.

Prerequisites

The following prerequisites apply before starting the monitoring tool:

- If the default embedded JMS server is used, then ALER needs to be running with the `cmee.eventframework.enabled` system setting set to true. This is to make sure that the JMS broker that is embedded within ALER is running so that the monitoring tool can connect to it and monitor the events.
- If an external JMS server is used, then the external JMS Server needs to be running and the JNDI-related eventing.properties that are required to connect to the external JMS server must be configured.

For more information, see "Configuring Connectivity Properties for External JMS Servers".

Usage

From a command prompt, run the Event Monitoring tool as follows:

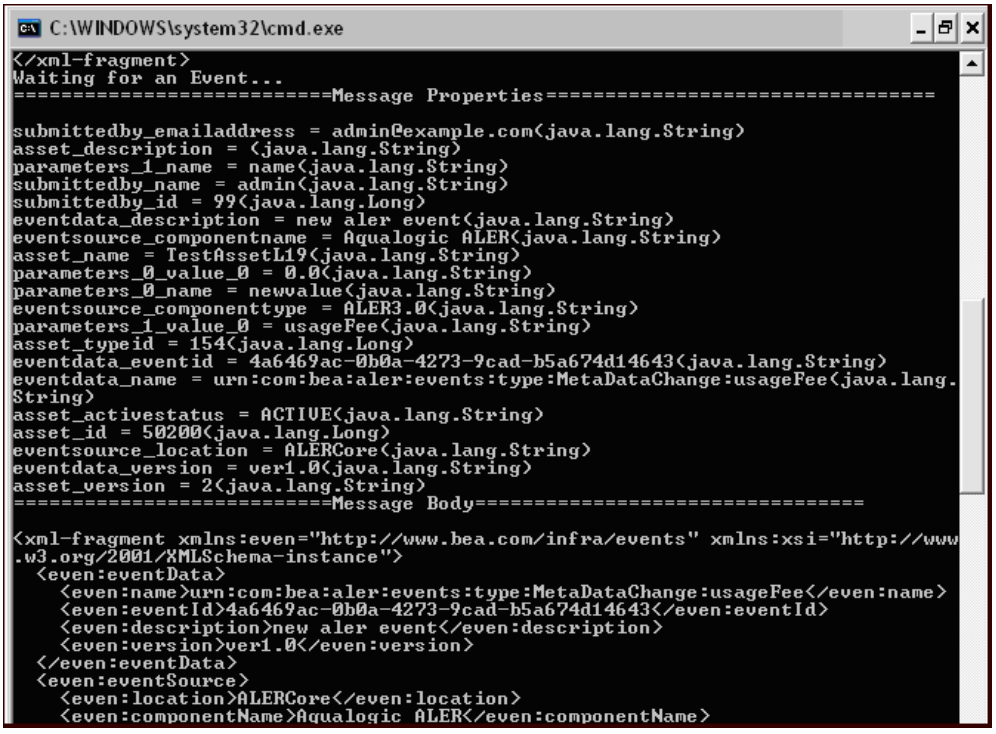
```
> event_monitor.bat <Path of WEB-INF\classes>
```

For example, if ALER is deployed to a domain named *alerdomain* under:
`D:\bea816\user_projects\domains\alerdomain`

Then the `<Path of WEB-INF\classes>` is:
`D:\bea816\user_projects\domains\alerdomain\applications\aler\aler-app\WEB-INF\classes`

This path is needed to get the JMS configuration from the `eventing.properties` file so that the tool can connect to the JMS server.

Figure 7-2 Event Monitor Console



Cleaning Up Stored Events

Sometimes it may be required to remove all the events that are stored by the Event Engine and also unsubscribe the durable subscription. The Event Cleanup tool can be used for this purpose.

Prerequisites

The following prerequisites apply before starting this tool:

- Set the ALER `cmee.eventframework.jms.subscribers.enabled` system setting to `false` so that the ALER Event Manager does not start the durable subscriber because this will be unsubscribed by the Clean Event tool.
- Restart ALER with the `cmee.eventframework.jms.subscribers.enabled` property set to `false`.

Usage

From a command prompt, run the Event Cleanup tool as follows:

```
> event_clean.bat <Path of WEB-INF\classes> <Name of Durable Subscriber>
<Message Selector>
```

For example, if ALER is deployed to a domain named *`alerdomain`* under:
`D:\bea816\user_projects\domains\alerdomain`

Then the *<Path of WEB-INF*`\classes>` is:
`D:\bea816\user_projects\domains\alerdomain\applications\aler\aler-app\WEB-INF\classes`

This path is needed to get the JMS configuration from eventing.properties so that the tool can connect to the JMS Server.

The *<Name of Durable Subscriber>* can be found in the `name` attribute inside the endpoint that requires event cleanup within the `EndPointEventSubscription.xml` as follows:

```
<sub:eventSubscription>
  <!--The name should be unique within this file since
  <sub:endPoint name="ALBPMEndpoint">
```

The *<Message Selector>* can be found in the `expression` attribute inside the endpoint that requires cleanup within the `EndPointEventSubscription.xml`

Note: The parameter can be omitted if the Message selector is not set or empty.

Sample Event Cleanup

Using the example above, navigate to the `workflow-tools` directory:

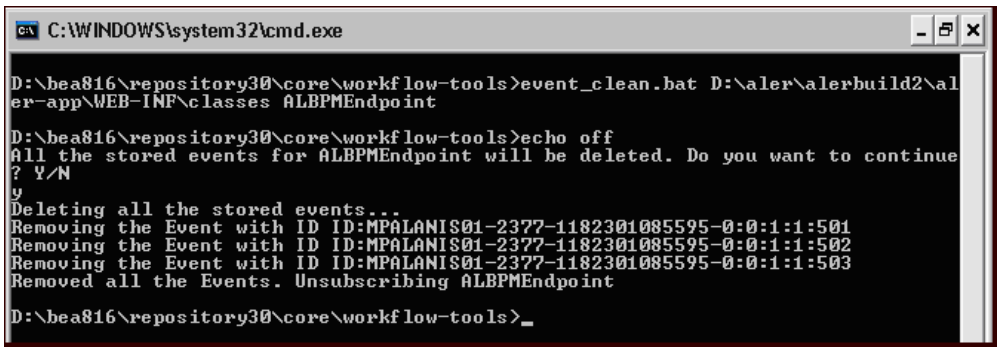
```
> cd D:\bea816\repository30\core\workflow-tools>
```

From the command prompt, type:

```
> event_clean.bat D:\aler\alerbuild2\aler-app\WEB-INF\classes
ALBPMEndpoint
```

The following is the output printed by the Event Cleanup tool to the console.

Figure 7-3 Event Cleanup Console



Generating the Workflow Config File

The Generate Workflow Config tool is used to generate the Workflow Configuration file (`workflow.xml`) by connecting to ALER. The tool populates the workflow.xml with configuration for asset types, categorizations, etc. by reading these entities from ALER. The Workflow Config file can then be customized as per your requirements. For example, you may need to configure and customize flows to add new asset types, projects, categorizations, etc.

For more information about configuring Advanced Registration Flows, see Chapter 5, “Configuring Advanced Registration Flows.”

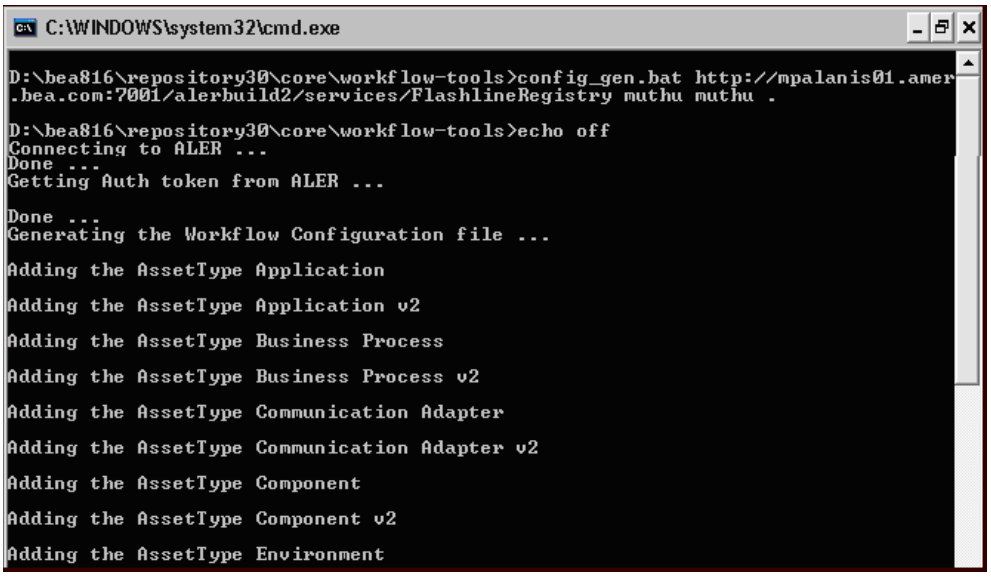
From a command prompt, run the Generate Workflow Config tool as follows:

```
> config_gen.bat URI User Password ConfigDir
```

where:

`URI` = ALER URI (for example: http://localhost:7001/alerbuild/services/FlashlineRegistry)
`User` = ALER user name
`Password` = ALER password
`ConfigDir` = the directory where the Config XML file will be created. If the file exists, it will be renamed to `workflow.xml.bak`.

Figure 7-4 Generate Workflow Configuration Tool



The `workflow.xml` file needs to be generated to the following directory:

```
<ALER Enterprise Edition Path>/server/<ALER Workflows
Project>/workflow.xml
```

Refreshing the Workflow Config File

The Refresh Workflow Config XML tool lets you to refresh a Workflow Config file without restarting the ALBPM Engine. For example, if the Workflow Config XML file is updated during development, running this tool allows the ALBPM Engine to use the updated version without restarting the engine.

Note: The ALBPM Engine must be running when running this tool.

From a command prompt, run the Refresh Workflow Configuration tool as follows:

```
> refresh_workflows.bat URI User Password
```

where:

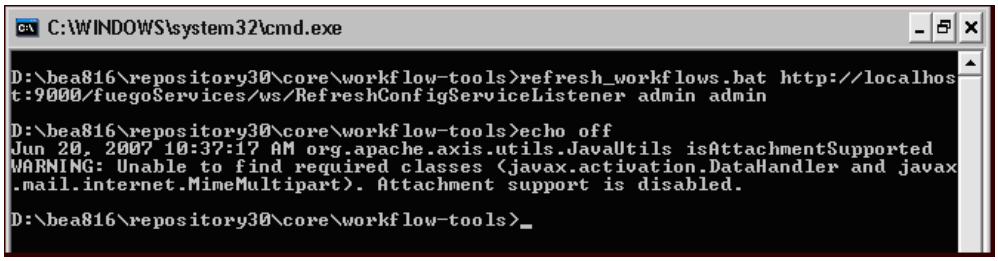
`URI` = ALBPM URI (for example, `http://localhost:9000/fuegoServices/ws/RefreshConfigServiceListener`)

`User` = ALBPM user name (for example, `aler_workflow_user`)

`Password` = ALBPM password (for example, `aler_workflow_user`)

Note: `aler_workflow_user` is created by the BEA Products Installer and is the default user that can be used with this tool.

Figure 7-5 Refresh Workflow Configuration Tool



Encrypting Your Passwords

For enhanced security, the Security Encrypt Password tool (`runWfSecurity.bat`) allows you to encrypt passwords that are stored in the Workflow Configuration and Subscription Service files.

From a command prompt, run the Security Encrypt Password tool as follows:

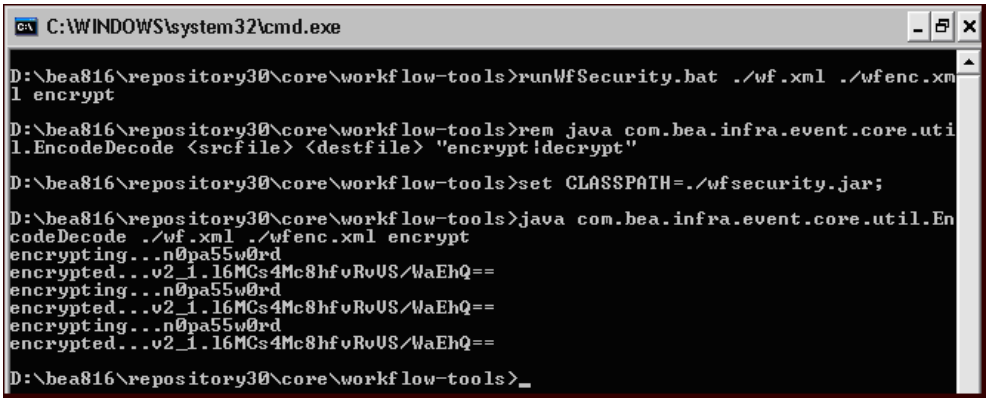
```
> runWfSecurity.bat srcFileName destFileName
```

where:

`srcFileName` = source config file with clear password.

`destFileName` = destination config file with decrypted password.

Figure 7-6 Security Encrypt Password Tool



CHAPTER 8

Extending the Event Manager for Web Service Endpoints

This section contains information on the following subjects:

- "Overview"
- "Developing a Web Service Endpoint"
- "Web Service Operations"
- "Developing a Notifier Plug-in"
- "Developing an Endpoint With an Incompatible Contract"

Overview

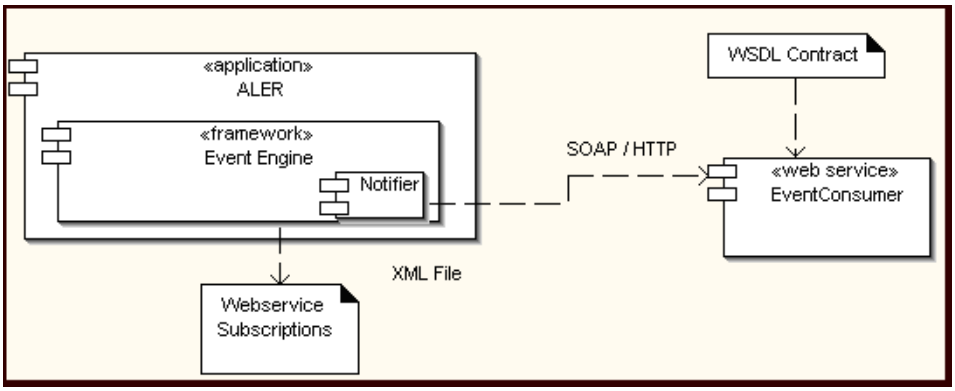
This document explains how to develop a new Web Service endpoint to consume the events that are emitted by the Event Manager and also explains how to extend the Event Manager to use other notifier plug-ins.

For information about configuring the Event Manager, see Chapter 3, "Configuring the ALER Event Manager."

Developing a Web Service Endpoint

The following figure shows how a Web Service endpoint can be plugged-in to receive the Events emitted by the ALER Event Manager.

Figure 8-1 Web Service Endpoint Plug-in



Following these steps to create a new Web Service endpoint and start getting events.

1. Pick up the WSDL contract defined by the Event Manager. This is bundled with the `eventNotifier.jar` located in the *<aler Webapp path>*`/WEB-INF/lib` directory.

2. Open the jar file and locate a WSDL named “EventListener.WSDL” and extract the WSDL to the file system. This WSDL is the abstract contract defined by the Event Manager and the new Web Service endpoint needs to implement the operation defined in the WSDL.

   Here is a snapshot of the WSDL file

Figure 8-2 Sample WSDL File

```
<?xml version="1.0" encoding="UTF-8" ?>
<!--  edited with XMLSpy v2007 sp2 (http://www.altova.com) by X Y (XYZ)   -->
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http://www.bea.com/aler/events/alerE
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:aler="http://w
  xmlns:ns="http://www.bea.com/infra/events" xmlns:ns1="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:wfStatus="http://www.bea.com/infra/events/workflow/status" name="AlerEventsListener"
  targetNamespace="http://www.bea.com/aler/events/alerEventsListenerWsdl/">
- <wsdl:types>
  - <xs:schema targetNamespace="http://www.bea.com/infra/events" xmlns:commonEvents="http://www.bea.com/infra/eve
      xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://www.w3.org/2001/XMLSchema" elementFormDefault="
      attributeFormDefault="unqualified">
    - <xs:annotation>
        <xs:documentation>Defines the common events structure</xs:documentation>
      </xs:annotation>
      <xs:element name="Event" type="commonEvents:EventType" />
    - <xs:complexType name="EventType">
      - <xs:sequence>
          <xs:element name="eventData" type="commonEvents:EventInfoType" minOccurs="1" maxOccurs="1" />
          <xs:element name="eventSource" type="commonEvents:EventSourceType" minOccurs="1" maxOccurs="1" />
          <xs:element ref="commonEvents:ExtendedData" minOccurs="0" maxOccurs="1" />
        </xs:sequence>
      </xs:complexType>
    - <xs:complexType name="EventInfoType">
      - <xs:sequence>
          <xs:element name="name" type="xs:string" />
          <xs:element name="eventId" type="xs:string" />
          <xs:element name="description" type="xs:string" />
          <xs:element name="version" type="xs:string" />
        </xs:sequence>
```

3. Complete the Web Service endpoint development using the tool or technology, as per the requirement. For example, you could develop a Proxy Service using AquaLogic Service Bus, which provides a feature where you can create a Web Service-based proxy service by pointing to a WSDL file. Make the Web Service running by completing the development of the Web Service.

4. Configure the Event Manager so that the Web Service endpoint's host, port, and URI, etc., are entered in the Subscription Manager file. For more information about configuring the Event Manager, see Chapter 3, "Configuring the ALER Event Manager."

5. Start ALER and trigger events using the Asset Editor and the Web Service endpoint will start getting the Events.

6. You can use the Event Monitoring tool that is bundled with ALER for debugging and monitoring the Events that are generated by the Event Manager.

Web Service Operations

This section describes the available operation for a new Web Service endpoint, and how to specify operations in the Event Manager.

Available Web Service Operations

The ALER Event Manager supports the following operations.

newEventRequestResponse

This operation takes the event object that is defined in the XML schema section as an input and returns the status as the output. The status is defined as string type. Additionally, if the status string starts with *Failure*, then the Event Manager will throw an exception and will try to re-deliver the event until it succeeds. If not, it will log the response and will deliver the next event unless there is a transport exception.

newEventRequestResponseString

This operation takes the event data in string form as an input and returns the status as the output. The status is defined as string type. Additionally, if the status string starts with Failure, then the Event Manager will throw an exception and will try to re-deliver the event until it succeeds. If not, it will log the response and will deliver the next event unless there is a transport exception.

newEventRequest

This operation takes the event object that is defined in the XML schema section as an input and is defined as a one-way operation.

newEventRequestString

This operation takes the event data in string form as an input and is defined as a one-way operation.

newEvent

This operation should be used only if the Process Engine is ALBPM. This operation internally invokes the `startSession` operation to start session to authenticate with ALBPM. It will also call `discardSession` after the invocation.

Selecting a Web Service Operation

The preferred Web Service operation can be selected by configuring the Event Manager's Subscription Manager the following way, as specified in the `operationName` element.

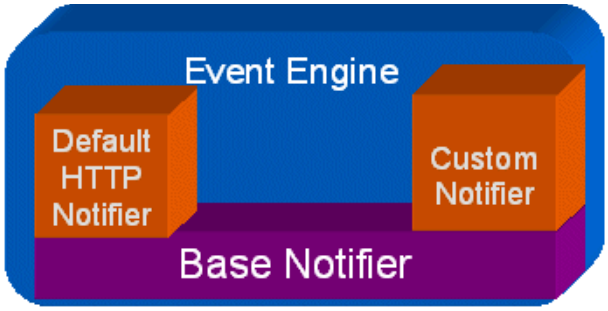
```
<sub:EventSubscriptionData
xmlns:sub="http://www.bea.com/infra/events/subscription"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <sub:eventSubscription>
    <sub:endPoint name="ALBPMEndpoint3">
      <sub:host>localhostt>
      <sub:port>9000</sub:port>
      <sub:uri>fuegoServices/ws/StatusChangeEnpointServiceListener</sub:uri>
     <sub:targetNamespace>http://www.bea.com/infra/events</sub:targetNamespace>
      <sub:operationName>newEvent</sub:operationName>
      <sub:authenticationData>
        <sub:basicAuthentication>
          <sub:username>admin</sub:username>
          <sub:username>admin</sub:username>
        </sub:basicAuthentication>
      </sub:authenticationData>
    </sub:endPoint>

<sub:notifierClass>com.bea.infra.event.notifier.plugin.http.DefaultHTTPEventNo
tifier    </sub:notifierClass>
    <sub:expression></sub:expression>
  </sub:eventSubscription>
</sub:EventSubscriptionData>
```

Developing a Notifier Plug-in

The ALER Event Manager includes a default SOAP/HTTP notifier. A new plug-in can be developed and plugged in if there are additional requirements, as illustrated here.

Figure 8-3 Notifier Plug-in



Follow these steps to make the new plug-in work with the Event Manager.

1. Develop a new Notifier Plug-in by extending the Java Class `AbstractEventNotifier` that is bundled with the ALER Event Manager. This class is bundled with the `eventNotifier.jar` located in the *<aler Webapp path>*`/WEB-INF/lib` directory. The `init()` and `sendNotification()` methods need to be overridden. Refer to the Javadoc for more information about these methods. The `handle()` method passes the event data in an XML Beans format, which can be used to send it to an external Web Service.

2. Configure the Subscription Manager file to point to the developed class. Modify the `notifierClass` element as follows:

```
<sub:EventSubscriptionData
xmlns:sub="http://www.bea.com/infra/events/subscription"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <sub:eventSubscription>
    <sub:endPoint name="ALBPMEndpoint3">
      <sub:host>localhost</sub:host>
      <sub:port>9000</sub:port>
      <sub:uri>fuegoServices/ws/StatusChangeEnpointServiceListener</sub:uri>
      <sub:targetNamespace>StuatusChangeEndpoint</sub:targetNamespace>
      <sub:operationName>newEvent</sub:operationName>
      <sub:authenticationData>
        <sub:basicAuthentication>
          <sub:username>admin</sub:username>
          <sub:username>admin</sub:username>
```

```
        </sub:basicAuthentication>
      </sub:authenticationData>
    </sub:endPoint>

<sub:notifierClass>com.bea.infra.event.notifier.plugin.http.DefaultHTTPEventNo
tifier</sub:notifierClass>
    <sub:expression>id &gt; 500</sub:expression>
  </sub:eventSubscription>
</sub:EventSubscriptionData>
```

3. Bundle the classes in a JAR file and copy it to `<aler Webapp path>/WEB-INF/lib` directory so that it is picked up by the classpath.
4. Restart the Event Manager and trigger an event using the Asset Editor.
5. The Event Manager will call the `init()` and `handle()` methods of the new notifier plug-in.

Developing an Endpoint With an Incompatible Contract

It is possible that there may be an endpoint with an Interface or Contract that is not compatible with ALER Event Manager. This is because the tool that is used to develop the endpoint may have restrictions to use the WSDL provided by ALER Event Manager, or there may be other inter-operability issues. The following approach can be used under those circumstances:

- Develop an event notifier plug-in to receive the event XML data and register with the Subscription Manager.
- Write the code in the new notifier plug-in that transforms the event data into the format that the remote Web Service expects.
- Invoke the remote Web Service by whatever API is supported by the remote endpoint.