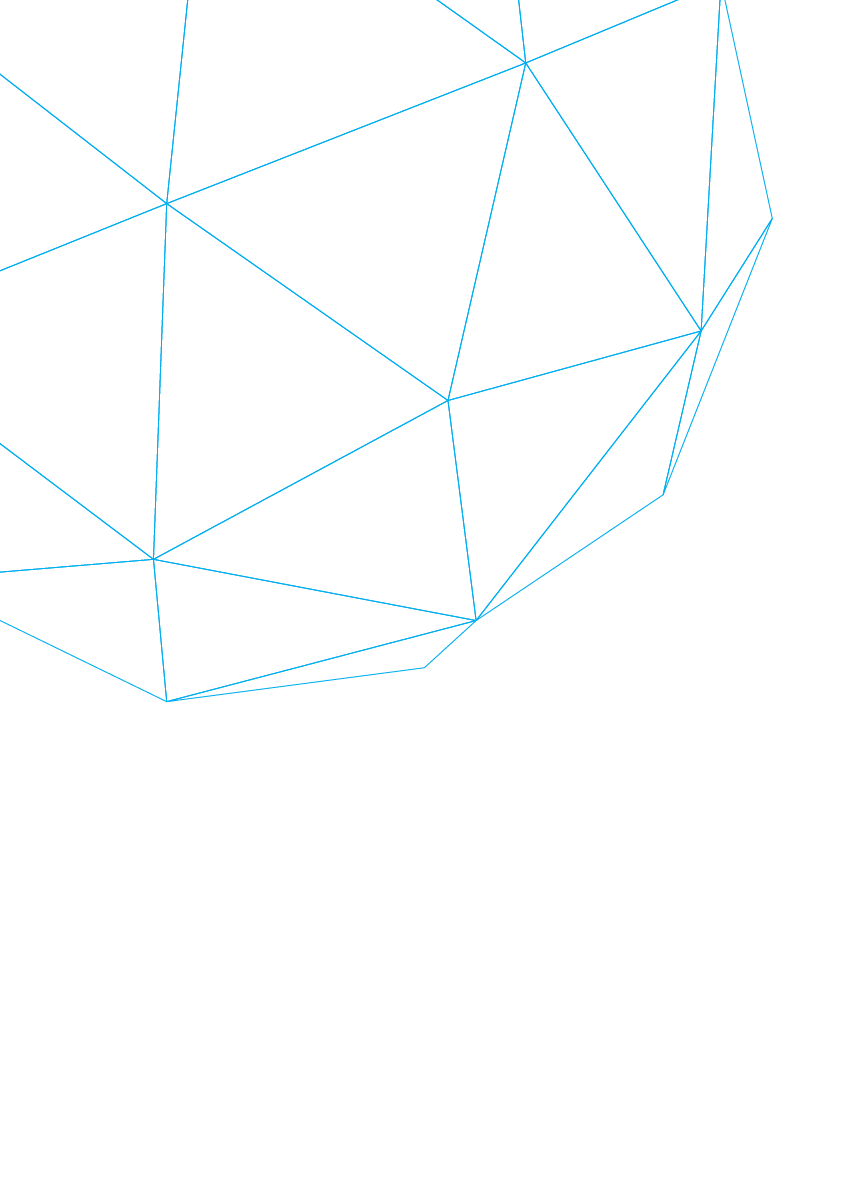
**UnionVms**

**Audit module integration**

Joakim Johansson



Audit module integration

This document describes how to integrate modules with the Audit module.

His document assumes that you have installed a local application server, postgres database and a JMS broker ( ActiveMq ).

It is also assumed that you have a schema “audit” in t postgres database and that you have ran the liquibase script provided in the svn repository for the Audit module.

To test your integration you need to deploy the Audit module and the Audit database module ( APP-ear and DB-ear ) located in /Modules/Audit/DB and /Modules/Audit/APP. For the deployments to work you need the prerequisites below fulfilled.

# Prerequisites

To be able to integrate with the Audit module there are some steps that has to be done before we start of writing code.

1. Add JMS queues for the Audit module
   1. UVMSAuditModel
   2. UVMSAudit
2. Add a datasource for the Audit module
   1. uvms\_audit

If you are Using Wildfly as your applicationserver the settings should look like below.

### JMS queues

<admin-object class-name=**"org.apache.activemq.command.ActiveMQQueue"**

jndi-name=**"java:/jms/queue/UVMSAuditModel"** use-java-context=**"true"**

pool-name=**"UVMSAuditModel"**>

<config-property name=**"PhysicalName"**>

**UVMSAuditModel**

</config-property>

</admin-object>

<admin-object class-name=**"org.apache.activemq.command.ActiveMQQueue"**

jndi-name=**"java:/jms/queue/UVMSAudit"** use-java-context=**"true"** pool-name=**"UVMSAudit"**>

<config-property name=**"PhysicalName"**>

**UVMSAudit**

</config-property>

</admin-object>

### Datasource

<datasource jta=**"true"** jndi-name=**"java:jboss/datasources/uvms\_audit"**

pool-name=**"uvms\_audit"** enabled=**"true"** use-ccm=**"true"**>

<connection-url>**jdbc:postgresql://127.0.0.1:5432/db71u**</connection-url>

<driver-class>**org.postgresql.Driver**</driver-class>

<driver>**postgresql**</driver>

<security>

<user-name>**audit**</user-name>

<password>**audit**</password>

</security>

</datasource>

## Maven dependency

As for all modules in the UnionVms system the only thing the integrating module has to do to integrate with another module is to import the model layer. So for Audit this means that the following dependency needs to be used

<dependency>

<groupId>**eu.europa.ec.fisheries.uvms.audit**</groupId>

<artifactId>**audit-model**</artifactId>

<version>**1.0.3**version>

<type>**jar**</type>

</dependency>

# How it works

Now that the prerequisites are met we can start with the actual integration. But first it is important to understand what happens when we are integrating the module. The following code shows an example on how to integrate with the Audit module.

**try** **{**

String auditData = AuditLogMapper.mapToAuditLog(“Asset”, “Create”, “{abc}”);

messageProducer.sendModuleMessage(auditData, ModuleQueue.AUDIT);

**}** **catch** **(**AuditModelMarshallException e**)** **{**

LOG**.**error**(**"Failed to send audit log message!**);**

**}**

The above example is a code snippet that shows how simple it is to integrate with the Audit module. Essentially what it does is to create a String that is an xml that will be sent over the JMS queue via a JMS message producer. If you are not using the standard Message producer provided by the Module archetype it is important that you use the object javax.jms.TextMessage when sending your message or the message cannot be parsed by the Audit module.

All modules comes with a predefined mapping class for the module a “ModuleRequestMapper”. This class handles all mapping needed to send requests and retrieve responses. The responses are located in a “ModuleResponseMapper”. In the case of the Audit module the Request mapper is called AuditLogMapper. In the mapper all needed methods available for the module are exposed.

So essentially what the integrating end needs to worry about is what to send, not how to map it or how the message structure needs to look like, all that is taken care of in the mapper.

## Arguments to the mapper

All arguments are generic so it is up to the module to decide what ends up in the audit

mapToAuditLog**(**String objectType**,** String operation**,** String affectedObject**)**

* objectType
  + This is the type of object affected that you want to log. If we take the Asset module as an example there are 2 entities that are affected in such a way that an audit log will be used. Assets and AssetsGroups. So depending on what object is affected the types are sent as “Asset” or “Asset group” as objectType
* Operation
  + This defines the operation that has happened and that you want to log. If we reuse the Asset and AssetGroup example there are 3 types of operations that are used “Create”, “Read” and “Update”. These are free text fields but it is recommended to use the strings “Create”, “Read” “Update”, “Delete” and “Archive” for basic CRUD operations. But it is up to the developer to decide what to send as oeration
* affectedObject
  + The affected Object if essentially an id that defines the object that is sent to the Audit log. This is used as a “GetById” field. For example In the Asset module the GUID identifier for the Asset is sent as the affectedObject when an Audit log is created. This GUID points to a historical Asset.

## GUI

In frontend you can read the audit logs from the rest interface from the audit module. There you have the ObjectType, what operation that was done to the ObjectType and an id that you can query for. For this to work properly All modules need to implement a “GetById” functionality so that the affectedObject can be queried and shown in the gui.

### Tip

In the asset module the mapping to the Audit module is made by an internal “Asset mapper” so that each individual methods for the objects is more easily accessed. See code below.

public class AuditModuleRequestMapper **{**

final static Logger LOG **=** LoggerFactory**.**getLogger**(**AuditModuleRequestMapper**.**class**);**

public static String mapAuditLogVesselCreated**(**String guid**)** **throws** AuditModelMarshallException **{**

**return** mapToAuditLog**(**AuditObjectTypeEnum**.**ASSET**.**getValue**(),** AuditOperationEnum**.**CREATE**.**getValue**(),** guid**);**

**}**

public static String mapAuditLogVesselUpdated**(**String guid**)** **throws** AuditModelMarshallException **{**

**return** mapToAuditLog**(**AuditObjectTypeEnum**.**ASSET**.**getValue**(),** AuditOperationEnum**.**UPDATE**.**getValue**(),** guid**);**

**}**

public static String mapAuditLogVesselGroupCreated**(**String guid**)** **throws** AuditModelMarshallException **{**

**return** mapToAuditLog**(**AuditObjectTypeEnum**.**ASSET\_GROUP**.**getValue**(),** AuditOperationEnum**.**CREATE**.**getValue**(),** guid**);**

**}**

public static String mapAuditLogVesselGroupUpdated**(**String guid**)** **throws** AuditModelMarshallException **{**

**return** mapToAuditLog**(**AuditObjectTypeEnum**.**ASSET\_GROUP**.**getValue**(),** AuditOperationEnum**.**UPDATE**.**getValue**(),** guid**);**

**}**

public static String mapAuditLogVesselGroupDeleted**(**String guid**)** **throws** AuditModelMarshallException **{**

**return** mapToAuditLog**(**AuditObjectTypeEnum**.**ASSET\_GROUP**.**getValue**(),** AuditOperationEnum**.**ARCHIVE**.**getValue**(),** guid**);**

**}**

private static String mapToAuditLog**(**String objectType**,** String operation**,** String affectedObject**)** **throws** AuditModelMarshallException **{**

**return** AuditLogMapper**.**mapToAuditLog**(**objectType**,** operation**,** affectedObject**);**

**}**

**}**