# RUS ACCOUNTING FOR UNICORE

Marcin Lewandowski

| | |
|---|---|
| Document Version: | 1.0.1 |
| Component Version: | 1.3.1 |
| Date: | 13 05 2011 |

# Contents

This is a RUS Accounting user manual providing information on running and using the accounting for UNICORE.

# 1 Introduction

This user manual covers UNICORE accounting system developed in ICM. The main modules of the system are:

- **rus-job-processor** - UNICORE job processor which allows to collect data from job, convert it into UsageRecord format and send via JMS. For UNICORE 6.3 use rus-job-processor in version 1.2.0 For UNICORE 6.4 use rus-job-processor in version 1.3.0+.
- **rus-service** - the extension which should be deployed as additional service on Unicore/X installation. The service maintains a database of usage records and provides a RUS (Resource Usage Service - OGF draft specification) interface. What is more the rus-service is consumer of records provided by rus-job-processor and rus-bssadapter.
- **rus-bssadapter** - the standalone daemon which is installed on Batch System (or LRMS) server. Usually this is the same machine where UNICORE TSI runs. It monitors accounting logs of the BSS and forward data via JMS.
- **rus-export-bat** - it's plugin for rus-service, which allows to export UR records via JMS using BAT format.
- **rus-ucc-plugin** - ucc plugin for quering rus-service
- **rus-ur-site** - web portal which allows presentation of gathered data
- **rus-usage-logger-feeder** - generates archival accounting data from Unicore log and sent to JMS queue (requires UsageLogger format from at least Unicore 6.4).

The rus-service merges data received from the rus-job-processor and from rus-bssadapter.

## 1.1 ActiveMQ Broker

The central point of data exchange is ActiveMQ broker. You can just download and extract it from ActiveMQ official web site. There is no need to perform any extra configuration. However you can add distinct uses with priviledges to write and read from queues.

---

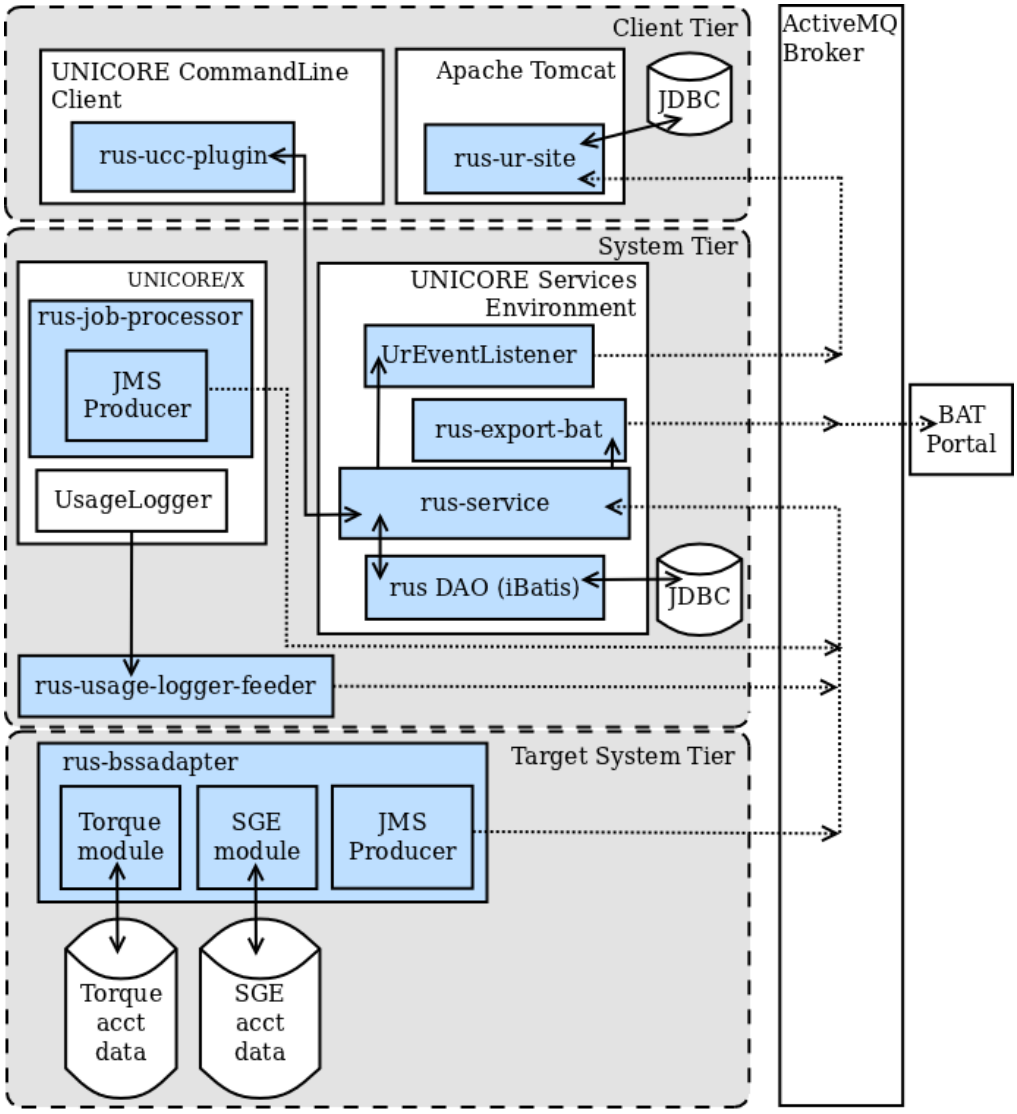
**Note**

**rus-export-bat** contains plugin which converts UsageRecords into BAT format, which in used as PL-Grid internal format to exchange data between UNICORE accounting and BAT portal. BAT portal which used in PL-Grid is written using GWT, and now it has almost nothing in common with Baltic Grid.

---

**Note**

**rus-service** and **rus-job-processor** can be deployed into same UNICORE server. However a good practise is to deploy them on separate servers. You just need two instances of UNICORE server. Job-processor needs to be installed on server, which accepts job requests. Server which hosts rus-server will be an auxiliary instance. After underlaying persistance layer of rus-server will contain lots of data, it can have small performance impact. However it was tested that having about 4,000,000 records in db has almost unnoticeable impact on performance.



# 2 Installation and configuration

## 2.1 rus-service

We assume here that `${unicorex}` is the installation directory of Unicore/X server.

1. Install accounting libraries in Unicore/X. The simplest way to achieve it is to copy the contents of distribution's `lib/` directory into `${unicorex}/lib`. However a good practice is to keep extensions' libraries in a separate directory. This involves bit more work:

   a. Create a new directory `${unicorex}/lib.extra`

   b. Copy the contents of distribution's `lib/` directory into `${unicorex}/lib.extra`.

   c. Modify `${unicorex}/bin/start.sh` to scan also `lib.extra` directory for libraries.
   To do so change the following lines:

   ```
   ...
   JARS=lib/*.jar
   CP=
   for JAR in $JARS ; do
   ...
   ```

   to:

   ```
   ...
   JARS=lib/*.jar
   JARS2=lib.extra/*.jar
   CP=
   for JAR in $JARS $JARS2 ; do
   ...
   ```

2. Add RUS service to `${unicorex}/conf/wsrflife.xml`.

   ```
   <property name="uas.security.accesscontrol.RUS" value="true"/>
   <service name="RUS" wsrf="false" persistent="false">
     <interface class="pl.edu.icm.unicore.accounting.commons.ws.ResourceUsagePort
     <implementation class="pl.edu.icm.unicore.accounting.service.ws.ResourceUsa
   </service>
   ```

3. Copy the contents of the distribution's `conf/` directory to the directory `${unicorex}/conf` (i.e. the rus directory which can be found there).

4. Configure ActiveMQ Broker properties in `${unicorex}/conf/rus/service.properties`. Additionally you can configure database properties in this file.

5. Append `pl.edu.icm.unicore.accounting.service.Bootstrap` to `uas.onstartup` property in `${unicorex}/conf/uas.config`. For example:

```
uas.onstartup=de.fzj.unicore.uas.util.DefaultOnStartup \
de.fzj.unicore.cisprovider.impl.InitOnStartup \
pl.edu.icm.unicore.accounting.service.Bootstrap
```

6. Configure authorization of the RUS service.
   You have two options prepared: to use role based authorization or to use DN-based authorization. Of course you can invent your own authorization scheme. The file rus-xaml-rules.xmlf contains example authorization rules for the role based authZ. One DN-based condition is commented. Note that the example XACML policy provides two entries: one for authorization of query operations (by default only allowed for users with *admin* role) and the latter for insertion of records. You can modify both.
   To install authorization rules paste the rus-xaml-rules.xmlf file contents to the `${unicorex}/conf/security_` file. The rules should be placed at the end of the file just BEFORE the last rule: `<Rule RuleId="FinalRule" Effect="Deny"/>`
   To use role based authorization set the role attribute of the UsageRecord provider identity to the value used in XACML policy (by default it is *bssaccounting*). Of course set it in the attribute store you use (XUUDB or UVOS or . . . ).
   If you want to use DN based authorization then replace role condition with DN condition and enter DN of your records provider identity.
   (Note: in default configuration records from rus-job-processor and rus-bss-adapter are fetched using JMS, so you don't have to configure XAMCL policy to allow records insertions).

7. Increase available memory in Unicore/X to at least 160MB. This setting is available in `${unicorex}/bin/start.sh`

```
# Memory for the VM
MEM=-Xmx128m
```

8. Restart Unicore/X.

---

**Logging**
In order to enable debug mode add following line to the `${unicorex}/conf/logging.properties`:

```
log4j.logger.pl.edu.icm.unicore.accounting=DEBUG
```

---

**Using MySQL instead of H2.**
There's possibility to use MySQL as storage instead of H2 (which is default setting). In order to change underlying DBMS:
Edit `${unicorex}/conf/rus/sqlconfig.xml`: Comment lines related to h2 and uncomment mysql section. Verify connection properties in: `${unicorex}/conf/rus/service.properties`.
Download and copy MySQL connector to `${unicorex}/lib/`
We support connectors in versions: 5.X.X+.

## 2.2 rus-job-processor

We assume here that `${unicorex}` is the installation directory of Unicore/X server.

1. Install accounting libraries in Unicore/X. The simplest way to achieve it is to copy the contents of distribution's `lib/` directory into `${unicorex}/lib`. However a good practice is to keep extensions' libraries in a separate directory. This involves bit more work:

   a. Create a new directory `${unicorex}/lib.extra`
   b. Copy the contents of distribution's `lib/` directory into ${unicorex}/lib.extra.
   c. Modify `${unicorex}/bin/start.sh` to scan also `lib.extra` directory for libraries. To do so change the following lines:

   ```
   ...
   JARS=lib/*.jar
   CP=
   for JAR in $JARS ; do
   ...
   ```

   to:

   ```
   ...
   JARS=lib/*.jar
   JARS2=lib.extra/*.jar
   CP=
   for JAR in $JARS $JARS2 ; do
   ...
   ```

2. Add job processor which accounts job stages to the `${unicorex}/conf/xnjs_legacy.xml` file. The following line must be added:

   ```
   <eng:Processor>pl.edu.icm.unicore.accounting.processor.AccountingJobProcessor
   ```

   into the section `<eng:ProcessingChain actionType="JSDL" ...>`. The line should be added as the last entry (typically after `<eng:Processor>de.fzj.unicore.xnjs.ems.process` entry).

3. Record merge is performed based on BSS Machine hostname. Make sure that property `CLASSICTSI.machine` in `xnjs_legacy.xml` equals to hostname setup by BSS Adapter. To override `CLASSICTSI.machine` property add `RUS.bssMachine` property in `xnjs_legacy.xml`. On BSS Adapter side you can manually alter this property by updating `rus.service.bssHostname` in `conf/rus_bssadapter.conf`. (equals is defined as string equal, so node113.domain.com NOT equals node113)

4. Configure JMS connection properties in `${unicorex}/conf/xnjs_legacy.xml`. You can add following properties:

```
<eng:Property name="RUS.PROCESSOR.jms.url" value="tcp://localhost:61616"/>
<eng:Property name="RUS.PROCESSOR.jms.queue" value="ur-parts"/>
<eng:Property name="RUS.PROCESSOR.jms.username" value=""/>
<eng:Property name="RUS.PROCESSOR.jms.password" value=""/>
<eng:Property name="RUS.PROCESSOR.jms.keystore" value=""/>
<eng:Property name="RUS.PROCESSOR.jms.keystore.password" value=""/>
<eng:Property name="RUS.PROCESSOR.jms.truststore" value=""/>
<eng:Property name="RUS.PROCESSOR.jms.truststore.password" value=""/>
```

   Only `jms.url` and `jms.queue` are required.

5. Restart Unicore/X.

---

**Logging**
In order to enable debug mode add following line to the `${unicorex}/conf/logging.properties`:

```
log4j.logger.pl.edu.icm.unicore.accounting=DEBUG
```

---

## 2.3 rus-bssadapter

1. Unpack the installation archive and place the contained folder in the proper destination on the same machine where Batch System Server (BSS) resides. Below we refer to the folder where the unpacked distribution resides with `${installPath}`

2. Update `${installPath}/conf/rus_bssadapter.conf` by setting paths, ActiveMQ Broker location and SSL configuration (which is optional)

3. Record merge is performed based on BSS Machine hostname. Make sure that property `CLASSICTSI.machine` in `xnjs_legacy.xml` equals to hostname setup by BSS Adapter. To override `CLASSICTSI.machine` property add `RUS.bssMachine` property in `xnjs_legacy.xml`. On BSS Adapter side you can manually alter this property by updating `rus.service.bssHostname` in `conf/rus_bssadapter.conf`. (Note: equals is defined as string equal, so node113.domain.com NOT equals node113)

4. Ensure that the user who will run BSS adapter have possibility to read accounting files in the directory specified by the `rus.service.torqueDataDir` property of the configuration file.

5. Usually you will want start the BSS adapter daemon to be started on the machine startup. Do it as your OS requires. Example initialization scripts can be found in `extra/init.d` directory of the distribution. Note that you shouldn't run this program as root.

6. Now you can start the server. Check logs if there are no errors.