RUS ACCOUNTING FOR UNICORE

Marcin Lewandowski, Krzysztof Benedyczak

Document Version:	1.1.0
Component Version:	2.0.0
Date:	20 05 2013

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 system is using JMS messaging as its internal transport layer and the OGF Usage Record (UR) specification as a data representation format.

Data about jobs is collected from a grid component (XNJS, which is a part of Unicore/X server) and from a batch system server (BSS). These pieces of information are merged together to form a full job record.

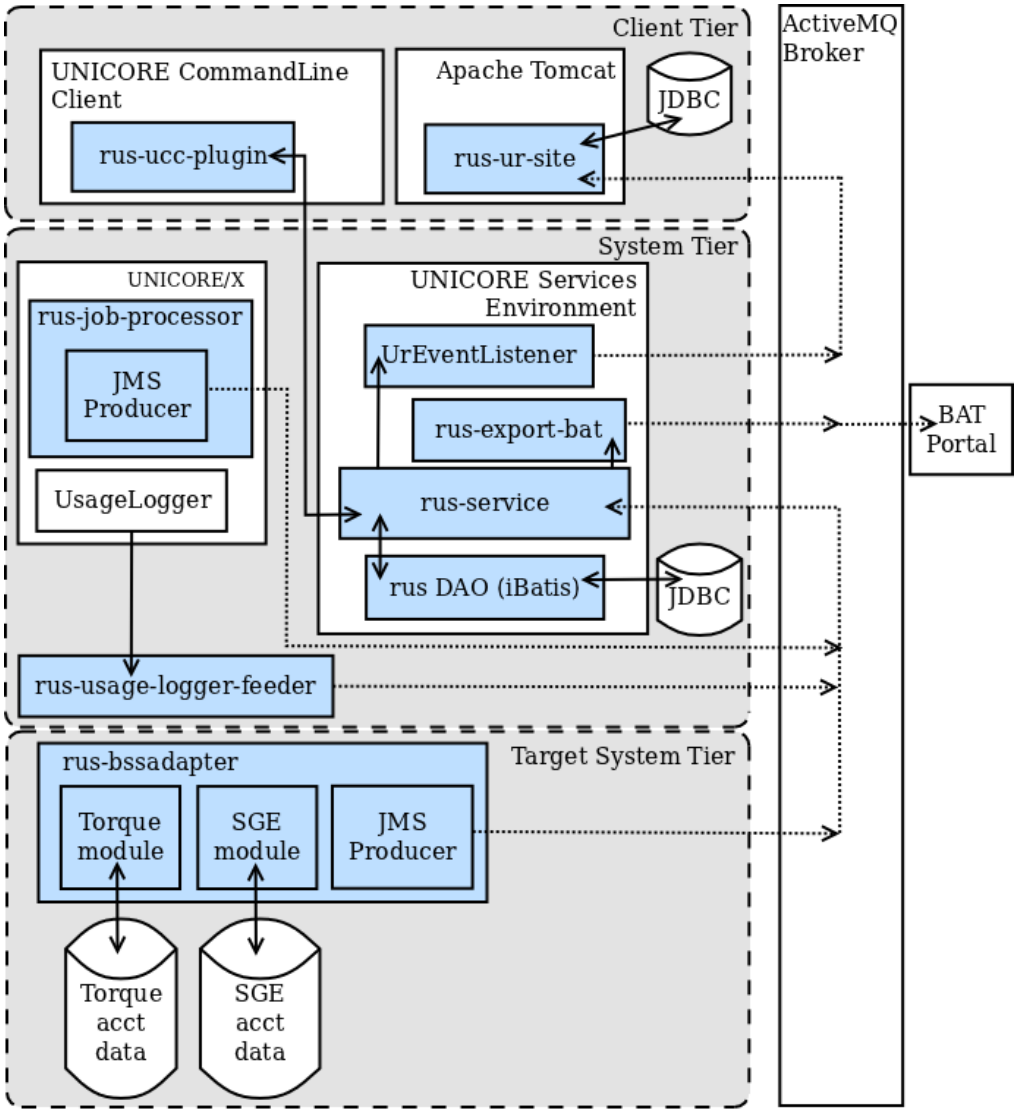
Please note that this accounting system is designed for production installations. Therefore it is fairly useless if you don't use a real BSS, but a Java or NOBATCH TSI.

The main modules of the system are:

- **rus-job-processor** - UNICORE job processor which allows for collecting grid-level information about jobs, converting it into UsageRecord format and sending via JMS to consumers (usually the rus-service).
- **rus-bssadapter** - a standalone daemon which is installed on a Batch System (or LRMS) server. Usually this is the same machine where UNICORE TSI runs. It monitors accounting logs of the BSS and forwards the collected data via JMS to consumers (usually the rus-service). Currently Torque, SLURM and Oracle/Sun GridEngine are supported.
- **rus-service** - Implementation of the main accounting service. It maintains a database of usage records and provides a Web Service management interface, based on OGF Resource Usage Service (RUS) draft specification. The rus-service is a consumer of records provided by the rus-job-processor and rus-bssadapter and merges them. The rus-service is distributed in two variants: as a standalone, ready to be used server and as a small bundle which can be added to an existing UNICORE container. Choose the one which suits your needs.
- **rus-export-bat** - it is a plugin for the rus-service. It allows for exporting UR records via JMS, using PL-Grid BAT format, to external consumers. PL-Grid BAT format, which in used in Polish NGI is a proprietary format and is rather unusable anywhere else. Note that PL-Grid version of BAT is a highly modified version of the original BAT system developed in the BalticGrid project.
- **rus-export-apel** - it is a plugin for the rus-service. It allows for exporting AUR records via Stomp to APEL. (https://wiki.egi.eu/wiki/APEL)
- **rus-ucc-plugin** - UCC client plugin for querying the rus-service. It is mostly useful for administrators to check the database contents and to perform some maintenance actions, e.g. to force execution of configured export plugins.

- **rus-webui** - (formerly: ur-site) web portal which allows for accounting data presentation. It provides a summaries, individual records and graphs presentation of the data with a vast amount of selection criteria and presentation modes.
- **rus-usage-logger-feeder** - it is a small additional utility, useful for administrators. Its function is to generate accounting records basing on an archival accounting data from Unicore/X usage log file(s). The generated accounting records are subsequently sent to the consumers via JMS (usually to the rus-service). The tool is especially useful when the accounting system is installed on a system which was already working for some time (and historical accounting data should be collected) or in case of long-lasting system failures (e.g. when the rus-job-processor can not send records to the JMS broker and drops some of them).

Additionally the ActiveMQ JMS broker must be installed, however this component is a generic, 3rd party server and it is not a part of the UNICORE RUS Accounting System distribution.



Accounting architecture: dotted lines: JMS communication, solid lines: inter-component communication.

1.1 Data model

The RUS accounting system collects information about individual jobs and stores it in the database. At the same time the daily summaries are stored in the database, using the information from the completed records.

The jobs table, providing the accurate and detailed information can quickly grow to an insane size. Therefore the records from this table are automatically moved to a historical table, after a

configurable period of time (e.g. 6 months). The historical table is not used by the system and its size does not influence the performance.

The aggregated reporting table holds only a subset of information of the summed amount of resources consumed by jobs performed on a particular site, by a particular user per day (to be precise: there are also other grouping variables). This table is subject to the same process of moving records to the historical table as we have in the case of individual jobs table. However as reporting table grows much slower then the jobs table, this operation can be done after a much longer period of time (e.g. 3 years).

The accounting system, since the 2.0 version uses the reporting data whenever possible, as its analyzes is significantly faster and at the same it provides majority of the data interesting for the stakeholders.

It is important to know that there are some (minor) limitations of the reporting data:

- The reporting table is using the start and end times of the job on an execution system to assign it to a particular day. Job submit/creation/queuing times are neither used nor visible. If a job spans several days, its resources consumption is distributed proportionally.
- Only the information about the already completed jobs is visible, jobs in progress are available only in the individual jobs table.
- Some failed or aborted jobs can be not included in reporting data, in particular those jobs which were aborted or failed before reaching the execution. This is by design, as such jobs have no start/end time which is used to assign a job to a proper reporting day. In future this limitation can be improved.

2 Compatibility

2.1 rus-service

Version 1.2.0 introduced new architecture so you can't use rus-service in version earlier than 1.2.0 with other components in version 1.2.0+.

If installing into an existing UNICORE service, as an add-on:

For UNICORE 6.4.x use rus-service 1.5.0

For UNICORE 6.5.0 use rus-service 1.6.0

2.2 rus-job-processor

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+.

UNICORE 6.5 has rus-job-processor installed (version 1.5.0+)

2.3 rus-usage-logger-feeder

If you want to use rus-usage-logger-feeder you need to have logs generated by Unicore 6.4.0+. Due to bug in Unicore/X 6.4.0 you can't collect information about user's VOs. This problem was fixed in Unicore 6.4.1.

Logs from earlier versions of UNICORE will be also correctly parsed, but they don't contain any useful information (we have information about a job, but we can't connect that information with its corresponding part from rus-bss-adapter).

2.4 rus-ucc-plugin

For UCC 6.5.0 use rus-ucc-plugin version 1.5.1

For UCC 6.4.2 use rus-ucc plugin version 1.5.0

2.5 rus-bss-adapter

We currently provide support for Torque and Sun Grid Engine. In order to choose BSS engine alter `rus.service.engine` property in `${bss-adapter}/conf/rus_bssadapter.conf`

3 JMS Notes

The accounting system is JMS based. It is good to be aware about the possible advanced settings like socket timeouts or low level logging. This can be achieved using the advanced settings of ActiveMQ clients which are used by the solution. The documentation is available available on ActiveMQ web pages. In particular the configuration of the popular TCP transport is here: http://activemq.apache.org/tcp-transport-reference.html

Note
Socket timeouts should be chosen with care. For instance too large values may significantly slow down export of records to external systems when even one of the receiving brokers is down. On the other hand too low values may cause problems in case of bad network conditions.

4 Installation and basic configuration

4.1 Directory layout

Currently the RUS accounting system is available only as a packed tar.gz archive. However it can be deployed on UNICORE installed both form the quickstart bundle (tar.gz or zip) and

from the RPM package. The following table summarizes locations of important files, which are dependent on the UNICORE installation method.

Name in this manual	tar.gz,zip	rpm	Description
CONF	<basedir>/conf/	**/etc/unicore/unicorex**	UNICORE/X config files
LIB	<basedir>/lib/	**/usr/share/unicore/unicorex/lib**	UNICORE/X Java libraries
LIB2	<basedir>/lib2/	**/usr/share/unicore/unicorex/lib2**	RUS Accounting extra java libraries
LOG	<basedir>/logs/	**/var/log/unicore/unicorex**	UNICORE/X log files
BIN	<basedir>/bin/	**/usr/sbin/**	start/stop scripts

4.2 Deployment planning

rus-job-processor is installed on each Unicore/X server which accepts user jobs. Similarly **rus-bssadapter** should be installed on a machine executing the jobs, i.e. the BSS server (usually the same where TSI is installed).

Location of the **rus-service** is up to the administrator. It can be deployed into the same Unicore/X server where the **rus-job-processor** is installed or even to a UNICORE server with other services like Registry. However a good practice is to deploy it on a separate UNICORE container. Such deployment introduces an increased maintenance effort as an additional server must be run, but allows for a better stability, ensures that problems in the **rus-service** won't influence a crucial Unicore/X server and splits the load.

Resource requirements of the rus-service are rather low. Tests have shown that on a commodity hardware, a database containing 1,000,000 of entries (jobs) behaves correctly and doesn't overload the machine. Around 2,000,000 of entries the commodity server can start to have problems. Note, however, that rus-service provides mechanism to solve this problem automatically by rolling older records - see the configuration section.

The client side, **rus-webui** and **rus-ucc-plugin** can be left as the last components (both are fully optional). The UCC plugin can be installed with any UCC installation, assuming the version matches (see the Compatibility section).

Installation should be started from deploying the ActiveMQ broker. Then the rus-service should be deployed. Finally all other components can be installed. Such installation order ensures that each installed component can be started and at least minimally tested without waiting for an installation of other components.

4.3 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 users with privileges to write and read from queues.

Example:

Let's suppose that we have the following installation:

- host grid_bss has installed Torque + bss-adapter.
- host grid_service has rus-service installed.
- host grid_processor has rus-job-processor installed.

We need one queue. Both grid_bss and grid_process will be producers, hence they need write access to queue. grid_service will be consumer.

```
<simpleAuthenticationPlugin>
  <users>
    <authenticationUser username="acct_bss" password=" ↩
       acct_bss_jms"
      groups="grid_producer,global"/>
    <authenticationUser username="acct_service" password=" ↩
       acct_service_jms"
      groups="grid_consumer,global"/>
    <authenticationUser username="acct_processor" password=" ↩
       acct_processor_jms"
      groups="grid_producer,global"/>
  </users>
</simpleAuthenticationPlugin>
```

And we configure queue access:

```
<authorizationPlugin>
  <map>
    <authorizationMap>
      <authorizationEntries>
        ...
        <authorizationEntry queue="chunks" read="admins, ↩
           grid_consumer" write="admins,grid_producer" />
        ...
      </authorizationEntries>
    </authorizationMap>
  </map>
</authorizationPlugin>
```

We use following credentials:

- on rus-bss-adapter: acct_bss/acct_bss_jms
- on rus-service: acct_service/acct_service_jms
- on rus-job-processor: acct_processor/acct_processor_jms

Of course it is also possible to set up SSL for the JMS messaging layer - see ActiveMQ documentation for details.

5 Configuration of the rus-service

In this section the **rus-service** is described in more details. RUS-service collects and stores accounting data - it is the heart of the whole system. RUS-service may also export accounting data to external systems, it can be used in a hierarchy and also can produce summary or aggregated reports.

5.1 Installation

You should follow one of the instructions below to install a standalone rus-service instance or to add it to an existing UNICORE server.

The 2.0.x version of the RUS service was tested on the 6.6.0 release of the base UNICORE distribution. Any other 6.6.x release should be equally fine. When installing into the existing container or when referring to the documentation of the base UNICORE container it is suggested to use this version.

5.1.1 rus-service - dedicated server instance

Note
Previously there was a dedicated tar.gz package, similar to the UNICORE quickstart package. This is not provided anymore and was replaced by the RPM version.

The easiest way to install the standalone version is to use the provided RPM package: unicore-rus-service. It is enough to download the package and install it normally using the `rpm -i <rpmfilename>` command.

You have to configure keystore/truststore (for instructions please refer to the UNICORE documentation of any UNICORE distribution as Unicore/X - the container's options are the same) Then go to point 6. under <<rus-service - installing into existing server>, which describes how to configure the rus-service.

Note
The dedicated rus-service server instance by default requires the working installation of the UNICORE Gateway (it is not included in the package). You can use a Gateway from another installation or reconfigure the UNICORE container hosting the RUS service not to use the Gateway (what is usually not suggested). Follow the standard UNICORE procedure to add a site to the Gateway.

5.1.2 rus-service - installing into existing server

The following instruction is valid for installation of tar.gz bundle into an existing UNICORE container e.g. UNICORE Registry or UNICORE/X server.

1. Install rus-service libraries into the server. The simplest way to achieve it is to copy the contents of distribution's `${LIB}` directory.

   However a good practice is to keep extensions' libraries in a separate directory. This involves bit more work: (a) create a new directory `${LIB2}`, (b) copy the contents of distribution's `lib/` directory into `${LIB2}`, (c) modify `${BIN}/start.sh` to scan also `${LIB2}` directory for libraries. To do so, after this line:

   ```
   ...
   CP=.$(find "${LIB}" -name *.jar -exec printf ":{}" \;)
   ...
   ```

   add an additional instruction, to load libraries from `${LIB2}` too (which must be defined first), as follows:

   ```
   ...
   LIB2="your extra jars path"
   CP=.$(find "${LIB}" -name *.jar -exec printf ":{}" \;)
   CP=${CP}:$(find "${LIB2}" -name *.jar -exec printf ":{}" ↩
       \;)
   ...
   ```

   `${LIB2}` variable can be alternatively defined in `${CONF}/startup.properties`.

2. Add RUS services to `${CONF}/wsrflife.xml`.

   ```
   <service name="RUS" wsrf="false" persistent="false">
     <interface class="pl.edu.icm.unicore.accounting.types. ↩
        ws.ResourceUsagePortType" />
     <implementation class="pl.edu.icm.unicore.accounting. ↩
        service.ws.ResourceUsageServiceImpl" />
   </service>
   ```

```
<service name="PluginsExecutor" wsrf="false" persistent ↩
   ="false">
  <interface class="pl.edu.icm.unicore.accounting. ↩
     pluginexecutor.service.PluginExecutorPortType" />
  <implementation class="pl.edu.icm.unicore.accounting. ↩
     service.ws.PluginExecutorServiceImpl" />
</service>
```

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

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

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

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

6. Configure authorization of the RUS service.
   Authorization is used for the rus-ucc-client, which allows users to query for records, based on a given criteria. If you intend to only allow users with role *admin* to use the service (what is a safe bet) you can skip this point. If you prefer to customize authorization of the RUS service access, you have two options prepared: to use a role based authorization or to use a DN-based authorization. Of course you can invent your own authorization scheme e.g. based on VO membership or other attributes. Provided example is configured to use the role based access, there is also a commented version for the DN-based authorization. Copy `doc/19rus-xacml2-policy.xml` into `${CONF}/xacml2Policies` directory. Review copied file and alter configuration to satisfy your requirements.

7. If you are installing rus-service to a UNICORE server installed from RPM/deb there are additional required steps.

   - define property for config location in `${CONF}/wsrflite.xml`. We refer to assigned value as `${RUSCONF}`

     ```
     <property name="rus.accounting.config" value="/etc/ ↩
        unicore/unicorex/rus/service.properties" />
     ```

   - edit `${RUSCONF}/service.properties`, update database url if you are using H2 and configuration file paths:

     ```
     accounting.db.jdbcUrl=jdbc:h2:/var/lib/unicore/ ↩
        unicorex/RUS
     ```

```
accounting.exportersConfigurationFile=/etc/unicore/ ↩
    unicorex/rus/rus_extensions.xml
accounting.reportingConfigurationFile=/etc/unicore/ ↩
    unicorex/rus/reporting.xml
```

8. Restart Unicore/X.

9. Verify your installation. Logs should contain:

```
2011-07-30 01:43:08,252 [main] INFO  ServiceConfigReader ↩
     - Running startup class <pl.edu.icm.unicore. ↩
   accounting.service.Bootstrap>
...
2011-07-30 01:48:58,882 [main] INFO  DefaultRUSManager   ↩
   - Using </etc/unicore/unicorex/rus/> as RUS  ↩
   configuration directory
2011-07-30 01:49:00,352 [main] INFO  DefaultRUSManager   ↩
   - RUS JOB consumer starting...
2011-07-30 01:49:00,366 [main] INFO  JMSFactory  - Using ↩
    JMS server: tcp://192.168.87.100:61616 queue:  ↩
   grid10
2011-07-30 01:49:00,416 [main] DEBUG JMSParticipant  -  ↩
   Using credentials: username[system], password ↩
   [******]
2011-07-30 01:49:00,427 [main] INFO  JMSParticipant  -  ↩
   RUS JMS Consumer started
2011-07-30 01:49:00,444 [JMS Consumer Timer] INFO   ↩
   JMSParticipant  - Connecting to jms: tcp ↩
   ://192.168.87.100:61616, queue name = grid10,  ↩
   username: system
2011-07-30 01:49:00,450 [RUS-Executor] INFO   ↩
   NotificationExecutor  - RUS Notification Executor  ↩
   started
```

Logging

In order to increase an amount of a logged information add the following line to the `${CONF}/logging.properties`:

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

Using MySQL instead of H2.
There's possibility to use MySQL as a storage instead of H2 (which is the default setting). In order to change the underlying DBMS:
Verify connection properties in: `${CONF}/rus/service.properties`. Comment lines related to H2 and uncomment the MySQL section.
Create an empty database, grant privilidges to user on created table and enter credentials to `${CONF}/rus/service.properties`.
Example DDL:

```
CREATE USER 'accounting'@'%' IDENTIFIED BY 'accounting_dba ↩
    ';
GRANT USAGE ON * . * TO 'accounting'@'%' IDENTIFIED BY ' ↩
    accounting_dba' WITH MAX_QUERIES_PER_HOUR 0  ↩
    MAX_CONNECTIONS_PER_HOUR 0 MAX_UPDATES_PER_HOUR 0  ↩
    MAX_USER_CONNECTIONS 0 ;
CREATE DATABASE IF NOT EXISTS `accounting` ;
GRANT ALL PRIVILEGES ON `accounting` . * TO 'accounting'@ ↩
    '%';
```

Note
When you install rus-service into UNICORE Registry server, some of the libraries (available for the Unicore/X) might be missing. For example: spring-core-3.0.5.RELEASE.jar and spring-expression-3.0.5.RELEASE.jar. Copy those jars from Unicore/X lib folder into UNICORE Registry lib folder.

5.2 Configuration options reference

The following options can be specified in the `rus/rus.service` configuration file:

Property name	Type	Default value / mandatory	Description
`accounting.exportersConfigurationFile`	string	`conf/rus/rus_extensions.xml`	Location of the file with the definitions of plugins used for exporting job records to external services.

Property name	Type	Default value / mandatory	Description
`accounting.factor.[.*]`	string *can have subkeys*	-	Can be used to provide centralized CPU time normalization settings. *.metric* defines metric, *.SITE.default* defines value for a SITE, *.SITE.host[xx-yy]* defines value for SITE's host range.
`accounting.jms.[.*]`	string *can have subkeys*	-	JMS properties are specified under this prefix. See the separate documentation.
`accounting.reportingConfigurationFile`	string	`conf/rus/reporting.xml`	Location of a file with definitions of plugins used for creating and possibly exporting reports on resource usage.
--- Database ---			
`accounting.db.dialect`	[h2, mysql]	`h2`	Database SQL dialect. Must match the selected driver, however sometimes more then one driver can be available for a dialect.
`accounting.db.driver`	Class extending java.sql.Driver	`org.h2.Driver`	Database driver class name. This property is optional - if not set, then a default driver for the chosen database type is used.
`accounting.db.jdbcUrl`	string	`jdbc:h2:data/RUS`	Database JDBC URL.
`accounting.db.password`	string	*empty string*	Database password.
`accounting.db.username`	string	`sa`	Database username.
--- Export executor ---			
`accounting.executor.maxFails`	integer >= 1	`25000`	Defines after how many tries a notification should be finally dropped.

Property name	Type	Default value / mandatory	Description
`accounting.executor.notifyStopThreshold`	integer number	`3`	If that many notifications for a particular export plugin fail in one round, then further notifications for this plugin are skipped. Set to negative value to disable this feature.
`accounting.executor.rescheduleInitial`	integer >= 1	`10`	How long (in s) to wait before retrying to send a notification.
`accounting.executor.rescheduleMax`	integer >= 1	`3600`	Defines the maximum retry time (in s) between retrying a failed notification.
`accounting.executor.rescheduleMultiplier`	integer >= 1	`2`	Defines how much the retry wait time should be multiplied, after each subsequent failure.
`accounting.executor.sleepTime`	integer >= 1	`500`	Sleep time (in ms) after one round of notification handling.
--- Old data maintenance ---			
`accounting.rolling.recordOlderThen`	string	`12m`	Defines how old job records should be moved to history. Leave unset to disable this feature. Values must be positive integers with one of the prefixes: y, m, d, h, s respectively for: years, months, days, hours or seconds.
`accounting.rolling.recordSchedule`	string	`0 35 4 * * ?`	Cron expression defining when the records rolling should be performed. See http://quartz-scheduler.org/documentation/quartz-2.x/tutorials/tutorial-lesson-06 for details.The basic format is: <*sec*> <*min*> <*h*> <*dayofMonth*> <*month*> <*dayOfWeek*> *[year]* and * is used for any value and ? for no specific value.

Property name	Type	Default value / mandatory	Description
`accounting.rolling.reportingOlderThen`	string	-	Defines how old reports should be moved to history. Leave unset to disable this feature. Values must be positive integers with one of the prefixes: y, m, d, h, s respectively for: years, months, days, hours or seconds.
`accounting.rolling.reportingSchedule`	string	`0 5 4 * * ?`	Cron expression defining when the reports rolling should be performed. See http://quartz-scheduler.org/documentation/quartz-2.x/tutorials/tutorial-lesson-06 for details.The basic format is: *<sec> <min> <h> <dayofMonth> <month> <dayOfWeek> [year]* and * is used for any value and ? for no specific value.

The JMS options are as follows. The same options are used to configure the JMS producers in the **rus-job-processor** and in the **rus-bssadapter**, however with other prefixes (`RUS.PROCESSOR.jms.` and `rus.bssadapter.jms.` respectively). Also the same options are used to configure JMS producers in the case of export plugins - this time with a simple prefix `jms.` (see below for examples).

Property name	Type	Default value / mandatory	Description
--- JMS messaging ---			
`accounting.jms.credential.[.*]`	string *can have subkeys*	-	Under this prefix the standard UNICORE properties can be used to configure a credential used for the JMS connection over TLS.
`accounting.jms.password`	string	*empty string*	Password used for username authentication to the broker.

Property name	Type	Default value / mandatory	Description
accounting.jms.queue	string	jobs	Name of a queue which should be used on the broker.
accounting.jms.truststore.[.*]	string *can have subkeys*	-	Under this prefix the standard UNICORE properties can be used to configure a truststore used for the JMS connection over TLS.
accounting.jms.url	string	tcp://localhost:61616	URL of the JMS broker.
accounting.jms.username	string	-	Optional username, which should be used for username authentication to the broker.

5.3 Configuring rus-service export plugins

In order to provide a possibility to extend rus-service module, data export plugins mechanism is available. Plugins allows you to invoke arbitrary action after a record is inserted or updated. Plugin configuration is in the ${CONF}/rus/rus_extensions.xml file. Plugin configuration can be changed on-the-fly. Module will automatically detect this event and reload configuration.

Sample configuration is shown below:

```
<extensions xmlns="http://www.icm.edu.pl/2010/rus-service/ ↩
   plugins/types">
  <extension class="pl.edu.icm.unicore.accounting.bat. ↩
     BATEventListener" id="bat">
    <config>
      <property key="jms.url">tcp://192.168.87.100:61616</ ↩
         property>
      <property key="jms.queue">jobs</property>
      <property key="bat.site.name">icm-unicore-testbed</ ↩
         property>
      <!-- Uncomment to enable login/password authentication ↩
          -->
      <property key="jms.username">system</property>
      <property key="jms.password">manager</property>
      <!-- Uncomment to enable SSL -->
```

```
      <!--
      <property key="jms.credential.format">jks</property>
      <property key="jms.credential.path">/path/to/your/ ↩
         keystore</property>
      <property key="jms.credential.password">123456</ ↩
         property>
      <property key="jms.truststore.type">keystore</property ↩
         >
      <property key="jms.truststore.keystorePath">/path/to/ ↩
         your/truststore</property>
      <property key="jms.truststore.keystorePassword ↩
         ">123456</property>
      -->
    </config>
    <condition>exitStatus != null and globalUserId != null</ ↩
       condition>
  </extension>
  <extension id="ur"
     class="pl.edu.icm.unicore.accounting.service.jms. ↩
        UrEventListener">
    <config>
      <property key="jms.url">tcp://192.168.87.100:61616</ ↩
         property>
      <property key="jms.queue">chunkes</property>
    </config>
    <condition>true</condition>
   </extension>
</extensions>
```

`Class` attribute inside `<extension>` element defines Java class name. Attribute `Id` allows you to use few plugins which are implemented by the same class. Id distinguishes different entries.

`Config` section settings are passed to a plugin as Java Properties.

In order to generate notification by notification manager, condition defined in `<condition>` element must be evaluated to `true`. See below for description how to create conditions.

We provide two plugins: `UrEventListener` (shipped inside `rus-service`) and `BATEventListener` (inside `rus-export-bat` module).

`UrEventListener` allows for simple passing of UsageRecords to another broker. It can be used to sent records to `rus-ur-site` module. Alternative usage is ability to create hierarchical structure of a grid. Records can be then gathered at several, arbitrary defined levels. Please keep in mind, that `<condition>` element allows you to filter records.

The second plugin (`BATEventListener`) converts record from UsageRecord format to BAT format, which is used inside the PL-Grid project.

Parameters responsible for broker communication are the same in case of the two plugins. `BAT EventListener` allows to use one additional configuration property: `bat.site.name`, which overrides the site for all records exported with this plugin.

Element `<condition>` have to contain a valid **Spring Expression Language** (SpEL) expression. The simple rules are quite intuitive for a detailed discussion of the SpEL syntax check the Spring documentation: http://static.springsource.org/spring/docs/3.0.x/reference/expressions.html#expressions-language-ref.

Sample SpEL expressions:

```
exitStatus != null and globalUserId != null
```

Matches all records that have exitStatus and globalUserId fields set. Such expression will match only complete records (with data from BSS and Unicore/X) of finished jobs.

```
exitStatus != null
  and globalUserId != null and
    not attributes['urn:SAML:voprofile:group']
    .?[#this.startsWith('/someVO')].isEmpty()
```

Matches all records as above, but the security related attributes of the job must possess the *urn:SAML:voprofile:group* attribute with the value starting with */someVO*. Note that the check is constructed in a way that correctly handles all situations when the attribute is not set at all.

Available elements in SpEL context:

element name	type	description	collected at
batchServer	String	**batch server name (machine name)**	**unicore and bss**
localJobId	String	**local job id**	**unicore and bss**
localUserId	String	**local user id**	**unicore and bss**
group	String	**user UNIX group**	**bss**
projectName	String	**project name**	**bss**
queue	String	**BSS queue name**	**bss**
ctime	Long	**time when job was created**	**bss**
qtime	Long	**time when job was queued**	**unicore and bss**
etime	Long	**time when job eligible to run**	**bss**
startTime	Long	**time when job was started**	**bss**
endTime	Long	**time when job was finished**	**bss**
execHosts	List<String>	**nodes which computed job**	**bss**
cpuTime	String	**cpu time**	**bss**

element name	type	description	collected at
wallTime	String	**wall time**	**bss**
exitStatus	Integer	**process exit status**	**bss**
globalJobId	String	**global job id**	**unicore**
vo	String	**virtual organization**	**unicore**
globalUserId	String	**global user id**	**unicore**
jobName	String	**job name**	**bss**
status	String	**job status**	**unicore and bss**
sitename	String	**site name**	**unicore**
attributes	Map<String, List<String>>	**security attributes**	**unicore**
origin	String	**origin of the record: bss unicorex merged or null**	**unicore and bss**

Programming custom plugins

If you want to create your own extension, you have to implement class which inherits from `AbstractDataEventListener`, which defines the following methods:

```
public void close();
public abstract boolean onData(UsageRecordType ur);
```

and contains a default constructor.

5.4 Configuring rus-service reporting

This feature was mainly developed to allow for generating Aggregated Usage Records, but can be extended to create other report types.

Configuration file is located in: `${CONF}/rus/reporting.xml`.

The configuration will be explained using a representative example:

```
<reporting xmlns="http://www.icm.edu.pl/2010/rus-service/ ↩
    plugins/reporting"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://www.icm.edu.pl/2010/rusa- ↩
         service/plugins/reporting rus-service-reporting.xsd">
  <reportItem id="APEL">
     <resultTransformerClass>pl.edu.icm.unicore.accounting. ↩
         service.reporting.AURReportingTransformer</ ↩
         resultTransformerClass>
     <cronExpression>0 * * * * ?</cronExpression>
```

```
    <postProcessorClass>pl.edu.icm.unicore.accounting. ↩
       service.reporting.JMSSendPostProcessor</ ↩
       postProcessorClass>
    <config>
      <property key="jms.url">tcp://192.168.87.100:61616</ ↩
         property>
      <property key="jms.queue">AUR</property>
      <property key="jms.username">system</property>
      <property key="jms.password">manager</property>
    </config>
    <whereAttributes>
      <globalUserId groupBy="true">
        <value>user1</value>
        <value>user2</value>
      </globalUserId>
      <localUserId>
        <any/>
      </localUserId>
      <machineName>
        <any/>
      </machineName>
      <host>
        <any />
      </host>
    </whereAttributes>
    <timePeriod>weekly</timePeriod>
  </reportItem>
</reporting>
```

We have one `reportItem` in this sample configuration file. Each of `reportItem` must have a unique id. Report items are used to distinguish between different report configurations.

The process of reporting consists of two steps: report preparation and report postprocessing. In the first step internal rus-service report data is formatted to the desired form. In the latter step the previously generated report for instance can be sent over network or saved to a file.

The `resultTransformerClass` entry defines what implementation will handle a translation from an internal format to the final report format. In the example the `pl.edu.icm.unicore.accounting.service.reporting.AURReportingTransformer` class performs a translation to the Aggregated Usage Record (AUR).

The `postProcessorClass` defines a class which performs the record postprocessing. The `pl.edu.icm.unicore.accounting.service.reporting.JMSSendPostProcessor` class sends generated reports via JMS.

The list of available implementations is:

Implementation class	Type	Description
`pl.edu.icm.unicore.accounting.service.reporting.AURReportingTransformer`	transformer	Produces OGF AUR draft specification reports
`pl.edu.icm.unicore.accounting.service.reporting.BypassReportingTransformer`	transformer	Do-nothing transformer. Mostly useful with Logging postprocessor when resulting format is not important.
`pl.edu.icm.unicore.accounting.service.reporting.LoggingPostProcessor`	postprocessor	Writes the record to the log file. You can log reports to a separate file by means of Log4j configuration. The logging is performed in INFO level with category equal to the postprocessor class name.
`pl.edu.icm.unicore.accounting.service.reporting.JMSSendPostProcessor`	postprocessor	Sends records over JMS channel.
`pl.edu.icm.unicore.accounting.car.CARReportingTransformer`	transformer	Produces report in CAR format. Available in rus-export-car package.
`pl.edu.icm.unicore.accounting.car.STOMPSendPostProcessor`	postprocessor	Sends records over STOMP channel. Makes sense only with CAR transformer. Available in rus-export-car package.

The `config` element contains arbitrary properties which are used to configure the chosen result transformer and postprocessor implementations. In the example the JMS configuration is set: it is possible to specify jms url, queue, username or password, and keystore/truststore information.

There are several advanced properties which can be set for all implementations. There is one of them which can be especially useful: `use.current.period`. If this property is defined and its value is `true`, then reporting data is collected from the current time period (day, week, month or year). Note that in this case subsequent invocations of the reporting in the same

period (e.g. in the same week) can produce different results, as new records can be added to the database in the meantime. When this option is not set, then reporting uses data from the last completed period of time. Example: Assuming that the report is generated on 25.02.2010, `timePeriod` is set to `monthly`, by default the report will contain all jobs accounted between 01.01.2010 and 31.01.2010. If additionally `use.current.period` is set to true, then the report will include the jobs accounted between 01.02.2010 and 25.02.2010.

The `cronExpression` element allows for setting up a schedule of the report generation. For information about the cron expression syntax refer to the page: http://www.quartz-scheduler.org/-documentation/quartz-1.x/tutorials/crontrigger.

The `timePeriod` element allows for selecting time range over which the report is generated. Available values: `daily`, `weekly`, `monthly`, `yearly`.

The optional `whereAttributes` element allows for filtering which records are used for generating report. Each subelement of the `whereAttributes` element must be one of the attributes enumerated in possible reporting xml values table. If you specify <any /> element as child of whereAttribute, than all values will be included in report. However, if you enumerate values, than only those enumerated values are used for generating the report.

Additionally each of the elements in `whereAttributes` can have the `groupBy="true"` XML attribute. If so, then the behaviour of the report generator is changed: a separate report is created for each and every distinct value of such an attribute (or for each distinct combination of values if more then one attribute has the `groupBy` set).

Example: We have grid system which is used by users: `CN=user1,OU=Grid`, `CN=user2,OU=Grid`, and some others. The following configuration generates two reports: one for user `CN=user1,OU=Grid`, and second for `CN=user2,OU=Grid` while for other users no report is generated.

```
<globalUserId groupBy="true">
  <value>CN=user1,OU=Grid</value>
  <value>CN=user2,OU=Grid</value>
</globalUserId>
```

But when we set `groupBy` to `false` (or simply skip it), then only one report will be generated, only based on jobs, which are submitted either by `CN=user1,OU=Grid` or `CN=user2,OU=Grid`.

```
<globalUserId groupBy="false">
  <value>CN=user1,OU=Grid</value>
  <value>CN=user2,OU=Grid</value>
</globalUserId>
```

Every child of `whereAttributes` element can have the `groupBy` attribute.

5.4.1 Available reporting xml attributes

Possible where attributes. You can use <any /> or enumerate values:

Attribute name	Description
globalUserId	DN of the user, always available
localUserId	local user identifier, always available
status	Status of the job, one of completed, failed, aborted, always available
machineName	Name of the BSS host (ususally the TSI machine), always available
queue	Name of the BSS queue, may be null
vo	Name of the job's VO, may be null
project	Name of the job's project, may be null
submitHost	Name of the Unicore/X host, may be null
site	Name of the site, may be null

Additionally each report transformer as its input receives the actual reported values, which are summed up for each entry (controlled by the groupBy attribute). The aggregated data consist of: number of jobs, walltime, CPU time, virtual memory and physical memory.

5.5 Configuring rus-export-car

With this plugin (which supersedes the older test version called rus-export-apel) one can export records from the RUS service using EMI Messaging Protocol for Accounting (defined: https://wiki.egi.eu/wiki/APEL/MessagingProtocol). Records are encoded in EMI Compute Accounting Record (CAR) format. It is possible to use this system either send individual job records or summary (aggregated) records.

In order to install rus-export-car, copy required jars from distribution into rus-service lib folder.

To configure the export of **aggregated records**, create new entry in reporting.xml:

```
<reportItem id="APEL-testendpoint">
        <resultTransformerClass>pl.edu.icm.unicore. ↩
           accounting.car.CARReportingTransformer</ ↩
           resultTransformerClass>
        <cronExpression>00 59 23 1 * ?</cronExpression>
        <postProcessorClass>pl.edu.icm.unicore.accounting. ↩
           car.STOMPSendPostProcessor</postProcessorClass>

        <config>
                <property key="jms.stomp.host">BROKER- ↩
                   HOSTNAME</property>
                <property key="jms.stomp.port">BROKER-PORT</ ↩
                   property>
                <property key="jms.stomp.user">BROKER-USER  ↩
                   or remove the property if not needed</ ↩
                   property>
```

```
                <property key="jms.stomp.password">BROKER- ↩
                   PASSWORD or remove the property if not  ↩
                   needed</property>
                <property key="jms.stomp.topic">BROKER-QUEUE ↩
                   </property>
                <property key="jms.stomp.responseTimeout"> ↩
                   BROKER-CONNECTION-TIMEOUT-MS</property>
                <property key="car.config.keystore.path"> ↩
                   KEYSTORE-WITH-PK</property>
                <property key="car.config.keystore.password ↩
                   ">KEYSTORE-PASSWORD</property>
                <property key="car.config.keystore.alias"> ↩
                   KEYSTORE-ALIAS</property>
                <property key="car.config.keystore.type">JKS ↩
                   </property>
                <property key="car.config.truststore.path"> ↩
                   TRUSTSTORE-WITH-RECEIVER-CERT</property>
                <property key="car.config.truststore. ↩
                   password">TRUSTSTORE-PASSWORD</property>
                <property key="car.config.truststore.type"> ↩
                   TRUSTSTORE-TYPE</property>
                <property key="car.config.truststore.alias"> ↩
                   TRUSTSTORE-ALIAS-OF-RECEIVER-CERT</ ↩
                   property>
                <property key="factor.default">DEFAULT-MULT- ↩
                   FACTOR</property>
                <property key="factor.metric">DEFAULT-METIRC ↩
                    (e.g. HEPSPEC)</property>
        </config>
        <whereAttributes>
                <globalUserId groupBy="true">
                        <any />
                </globalUserId>
                <site groupBy="true">
                        <any/>
                </site>
                <vo groupBy="true">
                        <any />
                </vo>
        </whereAttributes>
        <timePeriod>monthly</timePeriod>
</reportItem>
```

You have to set:

- broker settings where to send messages,

- keystore from which a private key is taken to **sign** records
- truststore from which a certificate of a consumer is taken to **encrypt** messages

Finally you can provide arbitrary filtering of records to be sent, but ensure to have `groupBy` turned on for `globalUserId`, `site` and `vo` - this is required by CAR specification. Also timePeriod must be set to `monthly` according to standard.

For general information about reporting refer to Configuring rus-service reporting.

To configure the export of **individual job records**, create new entry in `rus_extensions.xml`:

```
<extension class="pl.edu.icm.unicore.accounting.car. ↩
    SSMEventListener" id="apel-detailed">
        <config>
            <!-- see aggregated configuration - syntax is  ↩
                100% the same -->
        </config>

        <!-- condition for records selection can be freely  ↩
            set. Probably you want finished jobs with
              complete data from both LRMS and Unicore/X -->
        <condition>exitStatus != null  and  globalUserId !=  ↩
            null  and  batchServer != null</condition>
</extension>
```

The configuration of the broker, truststore, keystore and factors is the same as for the reporting configuration.

5.6 Accessing the rus-service with UCC

In order to install the UCC plugin:

1. Copy libraries from distribution `lib/` directory to `${ucc}/lib` or (preferrably) to the local user's UCC library folder, which is in `$HOME/.ucc/lib`.

This UCC extension adds new commands in the (new) Accounting category. Invocation of UCC with `help` option will provide a full list.

`Resource Usage Service` client allows you to query records, which meet specified criteria.

Examples:

```
./ucc acct-records -f status=completed,site=DemoSite https ↩
    ://grid02:8181/DEMO-SITE/services/RUS
  (get completed jobs from DemoSite)
```

```
./ucc acct-reports -f queue=ext,machineName=demo.server -g  ↩
   localUserName -t weekly -s DEMO-SITE
 (get weekly summaries of jobs grouped per uid, from queue  ↩
    ext on LRMS server demo.server)
```

UCC client can be used to schedule plugin notifications or report from given time period.

Examples:

```
./ucc acct-plugexec -i reporting:APEL -b 2011-11-01 -e  ↩
   2011-10-01 -s DEFAULT-SITE
(executes report with id 'APEL 'for time range: 2011-10-01  ↩
   to 2011-11-01)

./ucc acct-plugexec -i bat -b 2011-10-01 -e 2011-11-01 -s  ↩
   DEFAULT-SITE
(schedules 'bat' plugin execution for each record within  ↩
   time range, which meets
 criteria specified by the 'bat' plugin)
```

5.7 Records contents and merging

Note
This section provides advanced information, which is only relevant for developers and system integrators.

RUS-Service gets data from two sources per each job (or from one if one of the components is not configured): rus-job-processor with Grid data and bss-adapter with BSS data. Typically for each job, each of the data collectors produces multiple records, when job's state is changed. Rus-Service merges all records corresponding to a single job into a single record, which should contain the most recent job's status. Merging is done in different ways depending on record's element.

Records are matched together first by the globalJob id and then by localJobId and MachineName.

The following table shows all supported record elements along with components which can produce them and merging algorith being applied. Note that in many cases particular piece of data is provided only in *some* records comming from a particular source. For instance consumed memory is provided by bss-adapter but only in the record signaling job's end. More details (e.g. syntax of fields) can be found in https://www.ogf.org/documents/GFD.98.pdf

The merging algorithms used in the table:

- `first-win` - the first non-empty value is used, the subsequent ones are ignored.
- `bss-win-warn` - the value from BSS overwrites the older ones. When a different value is received a warning is produced, regardless of the actual change after merge (i.e. the value should be constant in the system).
- `bss-win` - the value from BSS overwrites the older ones.
- `ux-win-warn` - the value from Unicore/X overwrites the older ones. When a different value is received a warning is produced, regardless of the actual change after merge (i.e. the value should be constant in the system).
- `fail-on-change` - if a different value is received, then the new record is ignored with a warning (i.e. the value must be the same in the system)
- `ignore-new-warn` - if a different value is received, then the new value is ignored with a warning (i.e. the value should be the same in the system)
- `overwrite-status` - used to merge job status: the new status overwrite the old one, but only if the old one was not in one of the terminal states.

XML element name	**Source**	**Merging algorithm**	**Description**
`RecordIdentity@createTime`	BSS & U/X	see desc.	Creation time of the first record of the job. Upon merging the earliest value is always used (typically it is the value from the first record).
`RecordIdentity@recordId`	U/X	`first-win`	Unique record identity. In merged records it is the value of the first Id received.
`JobIdentity/GlobalJobId`	U/X	`fail-on-change`	UNICORE Job id
`JobIdentity/LocalJobId`	BSS & U/X	`fail-on-change`	BSS Job Id
`UserIdentity/LocalUserId`	BSS & U/X	`bss-win-warn`	Owner's uid
`UserIdentity/GlobalUserName`	U/X	`fail-on-change`	Owner's DN

XML element name	Source	Merging algorithm	Description
JobName	BSS	ignore-new-warn	Job's name
ProjectName	BSS	ignore-new-warn	Job's project (aka account string). In case of UNICORE jobs it is just the UNICORE job project.
Status	BSS & U/X	overwrite-status	Job's status
MachineName	BSS & U/X	fail-on-change	BSS server name
SubmitHost	BSS & U/X	ux-win-warn	Name of the machine from which job was submited to BSS. In records from Unicore/X it is its own hostname, in records from BSS it might be other hostname in case of non-UNICORE jobs.
Queue	BSS	bss-win	BSS queue name
Processors	BSS	ignore-new-warn	Total number of CPUs used on all nodes
NodeCount	BSS	ignore-new-warn	Number of nodes the job used
Host description= "CPUS=N; SLOTS=A,B..."	BSS	ignore-new-warn	One element for each used node contains its name. Additionally description provides number of CPUs at this host and list of occupied slots
StartTime	BSS	ignore-new-warn	Job's execution start time.
EndTime	BSS	ignore-new-warn	Job's execution end time.

XML element name	Source	Merging algorithm	Description
`WallDuration`	BSS	`ignore-new-warn`	Job's wall time
`CpuDuration`	BSS	`ignore-new-warn`	Job's CPU time
`TimeInstant type="etime"`	BSS	`ignore-new-warn`	Time when job was actually enqued by BSS
`TimeInstant type="qtime"`	BSS	`ignore-new-warn`	Time when job is ready to to be queued by BSS
`TimeInstant type="ctime"`	BSS	`ignore-new-warn`	Time when the job was first seen at BSS
`TimeInstant type="maxWalltime"`	BSS	`ignore-new-warn`	Start time + walltime, i.e. the latest point in time when the job should be finished. This is introduced for consistency checking.
`TimeInstant type="uxToBssSubmitTime"`	U/X	`ignore-new-warn`	Time when job was submitted to BSS from Unicore/X
`TimeInstant type="uxStartTime"`	U/X	`ignore-new-warn`	Time when it was detected by Unicore/X that the job was started. It is an approximation of the StartTime
`TimeInstant type="uxEndTime"`	U/X	`ignore-new-warn`	Time when it was detected by Unicore/X that the job was finished. It is an approximation of the EndTime
`Memory type="shared"`	BSS	`ignore-new-warn`	Virtual memory consumed by the job

XML element name	Source	Merging algorithm	Description
`Memory type="physical"`	BSS	`ignore-new-warn`	Physical memory consumed by the job
`Resource description="infrastructure"`	U/X	`ignore-new-warn`	Only filled with a constant value *unicore*
`Resource description="exit_status"`	BSS	`ignore-new-warn`	Exit status of the job
`Resource description="group"`	BSS	`ignore-new-warn`	Owner's gid
`Resource description="vo"`	U/X	`ignore-new-warn`	Owner's effective VO
`Resource description="sitename"`	U/X (BSS)	`ignore-new-warn`	Name of the whole site to which the Unicore/X belongs to. This may be configured also in bssadapter so records from BSS adapter can contain this value. However this is suggested ONLY in case when there is no RUS job-processor installed.
`Resource description="attributes"`	U/X	`ignore-new-warn`	Owner's authorization attributes including role and all VOs, Bencoded
`Resource description="recordOrigin"`	BSS & U/X	see desc.	Record's origin: `bss`, `unicorex` or `merged` if data was merged from both sources.

6 rus-job-processor

Since UNICORE 6.5.0, rus-job-processor is distributed together with Unicore/X server, therefore only its configuration is necessary. If you use older UNICORE distribution, then you should also use older release of this accounting software and check its documentation about installation

instructions.

1. Enable job processor which accounts job stages in the `${CONF}/xnjs_legacy.xml` file. The following line must be uncommented (or added if is not present):

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

   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.processors.UsageLogger</eng:Processor>` entry).

2. 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)

3. Configure JMS connection properties in `${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 ↩
       =""/>
   ```

   Only `jms.url` and `jms.queue` are required. If you want to confgure SSL for JMS connections, it is done using the same properties for credential and truststore as in any other UNICORE component (starting from 6.6 release). It is only required to use the `RUS.PROCESSOR.jms.` prefix. Full reference is available here

4. Restart Unicore/X.

5. Verify your installation. **Submit any job**. If logging is set to DEBUG mode (see note) log file should contain information about a sent record.

Logging

In order to enable debug mode add following line to the `${CONF}/logging.properties`:

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

Submit host property
UR element <submitHost> has default value of the host's canonical name. If you want to override this value setup following property in `${CONF}/xnjs_legacy.xml`:

```
<eng:Property name="RUS.ce.node.name" value=" ↩
    your_CE_hostname"/>
```

Sitename property
UR element <resource description="sitename"> has default value set to <machineName>. If you want to override this value setup following property in `${CONF}/xnjs_legacy.xml`:

```
<eng:Property name="RUS.site.name" value="your_site_name ↩
    "/>
```

7 rus-bssadapter

Installation and configuration is relatively simple:

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}`. Alternatively install the package from the RPM.

2. Update `${installPath}/conf/rus_bssadapter.conf` by setting paths, ActiveMQ Broker location and SSL configuration (which is optional). Choose a correct batch subsystem. In case of each of them there are only few extra options to be set. The configuration file is simple and well commented. The configuration options reference is provided below.

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 data. In case of SGE and Torque it means the directory specified by the `rus.service.accountingDataDir` property of the configuration file. In case of SLURM `sacct -a` must show all jobs to the BSS user.

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.

The detailed configuration options reference follows:

Property name	**Type**	**Default value / mandatory**	**Description**
--- Batch subsystem settings ---			
`rus.bssadapter.accountingDataDir`	filesystem path	`/var/spool/torque/server_priv/accounting/`	Path to accounting data, used for Torque and SGE.
`rus.bssadapter.sacctCmd`	string	`sacct`	Path of the sacct program
`rus.bssadapter.sacctExtraArgs`	string	*empty string*	Don't touch this unless you know what you are doing! Allows to pass additional switches to the sacct application, e.g. to limit what is accounted. Note that the program adds a lot of switches on its own, and you shouldn't interfere with them. The actual list is printed in the DEBUG mode.
`rus.bssadapter.slurmMaxWalltimeHours`	integer number	`100`	The maximum time limit of of the partition allowing for the longest jobs, in hours. Note that this need not to be precise - it is used only in rare cases, to provide an approximate maximum job's finish time, when no other information is available.
--- General settings ---			

Property name	Type	Default value / mandatory	Description
`rus.bssadapter.bssHostname`	string	-	Name of the local hostname. If not specified the value will be the automatically resolved local host name
`rus.bssadapter.dataPointerFile`	string	`conf/dataPointer.txt`	File holding information about the progress of sending accounting data. It is created and filled automatically.
`rus.bssadapter.engine`	[torque, sge, slurm]	`torque`	BSS engine to which is used
`rus.bssadapter.jms.[.*]`	string *can have subkeys*	-	Under this prefix the JMS connection settings needs to be provided (see separate documentation).
`rus.bssadapter.normalizationFactor`	floating point number	-	Value of the normalization factor of the normalization metric. If undefined, then metric information won't be put into the records produced by this component.
`rus.bssadapter.normalizationMetric`	string	-	Name of the normalization metric, such as HEPSPEC. If undefined, then metric information won't be put into the records produced by this component.
`rus.bssadapter.sitename`	string	-	Optional property, when set its value is used to provide the sitename to which this BSS system belongs to. This is useful only in case when there is no Grid (Unicore/X) RUS job processor installed for this BSS, as normally it provides this information.

Property name	Type	Default value / mandatory	Description
`rus.bssadapter.sleepLongTime`	integer number	`180`	Time between subsequent accounting data scans (in seconds) used if at least last 3 scans finished with an error.
`rus.bssadapter.sleepTime`	integer number	`30`	Time between subsequent accounting data scans, in seconds.

The detailed JMS properties reference is available in the rus-service configuration section.

8 Installation of rus-site

Ur-site requires own database. We copy data from rus-site via JMS using `pl.edu.icm.unicore.accounting.service.jms.UrEventListener` plugin. We can specify condition, to filter outgoing traffic.

Look at architecture overview for better understaning data flow:

rus-service → UrEventListener → (JMS) → broker → (JMS) → ur-site.

Following informations describes installation on Apache Tomcat. For other containers some of paths may be different.

Additionally we assume that we deploy application under /ur-site context.

1. Download and install Tomcat.
2. Create JNDI config in: `${tomcat}/conf/Catalina/localhost/ur-site.xml` (if you'are using ur-site as context).

```
<Context path="/ur-site"  debug="5" reloadable="true"  ↩
   crossContext="false">
  <Resource name="jdbc/ur-site" auth="Container" type=" ↩
     javax.sql.DataSource"
     factory="org.apache.commons.dbcp. ↩
        BasicDataSourceFactory"
     url="jdbc:mysql://locahost/RUS"
     driverClassName="com.mysql.jdbc.Driver"
     username="rus" password="rus_dba"
     maxIdle="4" maxActive="20" />
  <Resource name="jms/connection" auth="Container" type ↩
     ="org.apache.activemq.ActiveMQConnectionFactory"
```

```
      description="JMS Connection Factory"factory="org. ↩
         apache.activemq.jndi.JNDIReferenceFactory"
      brokerURL="failover:tcp://localhost:61616"  ↩
         brokerName="ActiveMQBroker" userName="system"  ↩
         password="manager" />
  <Resource name="jms/destination" auth="Container" type ↩
     ="org.apache.activemq.command.ActiveMQQueue"
     description="UR queue" factory="org.apache.activemq ↩
        .jndi.JNDIReferenceFactory" physicalName="ur- ↩
        site"/>
  <Resource name="app/config" auth="Container"  ↩
     description="properties file location" type="java. ↩
     lang.String"
     factory="pl.edu.icm.unicore.accounting.site.jndi. ↩
        StringFactory"
     value="file:///home/ml054/ur-site/ur-site. ↩
        properties"  />
</Context>
```

Properties file has following content:

```
container.auth.enabled=true  #enabled by default
uvos.auth.enabled=false  #disabled by default

uvos.auth.entry.point=https://uvos-server:2443/webauth/ ↩
   VOauthentication.do
uvos.auth.issuer=http://localhost/ur-site/

uvos.query.server=https://uvos-server:2443
uvos.query.admin.bss=bss_admin@/grid
uvos.query.user.bss=bss_user@/grid

uvos.auth.keystore=/path/to/your/keystore
uvos.auth.keystoreAlias=mykey
uvos.auth.keyPasswd=123456
uvos.auth.keystorePasswd=123456
uvos.auth.keystoreType=JKS
uvos.auth.truststore=/path/to/uvos/PUBLIC/KEY
uvos.auth.truststoreType=JKS
uvos.auth.truststorePasswd=123456
```

3. Create ur-site database with following DDL:

```
CREATE TABLE IF NOT EXISTS `RECORDS` (
`BSS_HOST` varchar(128) DEFAULT NULL,
```

```
  `BS_ID` varchar(255) DEFAULT NULL,
  `ACTION_UUID` varchar(36) DEFAULT NULL,
  `ID` int(11) NOT NULL AUTO_INCREMENT,
  `RECORD` text NOT NULL,
  `STATUS` varchar(20) DEFAULT NULL,
  `GLOBAL_USER_ID` varchar(255) DEFAULT NULL,
  `QUEUE` varchar(100) DEFAULT NULL,
  `SUBMIT_HOST` varchar(255) DEFAULT NULL,
  `START_TIME` datetime DEFAULT NULL,
  `END_TIME` datetime DEFAULT NULL,
  `QUEUED_TIME` datetime DEFAULT NULL,
  `VO` varchar(128) DEFAULT NULL,
  PRIMARY KEY (`ID`),
  UNIQUE KEY `BSS_HOST` (`BSS_HOST`,`BS_ID`),
  UNIQUE KEY `ACTION_UUID` (`ACTION_UUID`),
  KEY `IDX_STATUS` (`STATUS`),
  KEY `IDX_GLOBAL_USER_ID` (`GLOBAL_USER_ID`),
  KEY `IDX_QUEUE` (`QUEUE`),
  KEY `IDX_SUBMIT_HOST` (`SUBMIT_HOST`),
  KEY `IDX_START_TIME` (`START_TIME`),
  KEY `IDX_END_TIME` (`END_TIME`),
  KEY `IDX_QUEUED_TIME` (`QUEUED_TIME`),
  KEY `IDX_VO` (`VO`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ↩
   ;
```

4. Deploy war into container.
5. Install MySQL Connector to Tomcat.
6. Copy `xmlsec` and `commons-logging-api` libraries (jar files) to the `endorsed/` directory of Tomcat server. You can find those libraries (in correct version) in the ur-site web application's `WEB-INF/lib` folder. After this operation restart of the Tomcat server is required.
7. In order to use container autorization user requires `ur_site_admin` role. Ur-site calls `container-auth.html`, which is protected by container authorization. You can use both uvos and container authorization.

   Example Tomcat `CATALINA_HOME/conf/tomcat-users.xml`:

```
<tomcat-users>
  ...
  <role rolename="ur_site_admin" />
  <user username="ur_site_admin" password="Ad|\/|!|\|"  ↩
     roles="ur_site_admin"/>
  ...
</tomcat-users>
```

8. (optional) You can also configure authentication using UVOS. You need UVOS server (http://uvos.chemomentum.org/) + webauth plugin installed to UVOS. Information about installation and configuration of UVOS + webauth plugin are avaiable on UVOS home page.

   Configure attributes required to admin and user role in ur-site.

   `uvos.query.admin.bss=bss_admin@/grid` means that ADMIN must have `bss` attribute, with value = `bss_admin` in group `bss_admin`. So general form is: `uvos.query.(admin|user).attributeName=attributeValue[@optionalGroup]`.

Note
Rus-ur-site requires seperate database then rus-serivce.

Note
Ur-site requires active connection to broker to start!

Note
`uvos.auth.truststore` contains UVOS public key, **NOT** public keys of trusted CA.

Logging of the ur-site web application is controlled by a file `WEB-INF/classes/log4j.properties`. By default it logs to a separate file: `CATALINA_HOME/logs/ur-site.log`. You can reconfigure logging settings there.

9 Usage Logger Feeder

When infrastructure on which we installing accounting system works for some period of time, then the question arrives: Is there any possibility to import archival accounting data to rus-accounting service? The answer is true, but we need archival log files from Unicore/X in version 6.4.1+. Those logs contains UsageLogger entries. Based on those entries we can generate accounting data, which are identical to those which are generated by `rus-job-processor`.

Rus-usage-logger-feeder is a standalone application. It is configured with ActiveMQ broker parameters. As input parameters we supply time range of imported data and path to the Unicore/X server logs. Application search throught input files in order to find UsageLogger entries. Every item found is parsed and UsageRecord is generated, and sent to the broker. After the import operation is completed, the application displays information about amount of records found per each day.

Sample execution:

```
[ml054@raptor bin]$ ./run.sh 2010-10-23 2010-10-29 /home/ ↩
   ml054/sample_logs/uas.log
Using following parameters:
Start date : 2010-10-23
End date : 2010-10-29
Log filename: /home/ml054/sample_logs/uas.log
2011-04-11 12:05:49,913 [main] INFO
pl.edu.icm.unicore.accounting.commons.jms.JMSFactory
- Using JMS server: tcp://localhost:61616 queue: feeder
Processing file: /home/ml054/sample_logs/uas.log.2010-10-23
...
Processing file: /home/ml054/sample_logs/uas.log.2010-10-29
------------------------------------------------------------------------ ↩

---- OPERATION COMPLETED ---------------SUMMARY ↩
   :-----------------------
------------------------------------------------------------------------ ↩

# of Items + File name
-----------+------------------------------------------------------------ ↩

243        + uas.log.2010-10-23
234        + uas.log.2010-10-24
1022       + uas.log.2010-10-25
562        + uas.log.2010-10-26
706        + uas.log.2010-10-27
246        + uas.log.2010-10-28
1011       + uas.log.2010-10-29
```

10 Troubleshooting

10.1 Common problems

This sections describes problems caused by broker interaction.

10.1.1 Unable to connect to broker:

Log:

```
2011-07-30 08:15:13,537 [JMS Consumer Timer] INFO   ↩
   JMSParticipant  - Rescheduling JMS connection...
```

```
2011-07-30 08:15:23,538 [JMS Consumer Timer] INFO   ↩
   JMSParticipant  - Connecting to jms: tcp ↩
   ://192.168.87.100:61616, queue name = grid10, username:  ↩
   system
2011-07-30 08:15:23,554 [JMS Consumer Timer] INFO   ↩
   JMSParticipant  - Rescheduling JMS connection...
```

Solution:

Enable debug mode in logging.properties for more info about cause. Than check your connection with broker.

10.2 rus-service

For ActiveMQ problems please refer to common problems section.

10.2.1 Notifications subsystem:

When scheduled notification fails for 25000 times, then following message is displayed:

```
Notification with id = <notification-id> for plugin: <plugin ↩
   -id> failed for XXXX times. Removing...
```

and notification is removed.

When plugins configuration is changed but there are remaining notifications for non-existent plugin, than following message is shown:

```
Deleting queued notification for not existent plugin: < ↩
   plugin-id>
```

When notification fails for 3 times in single notification executor interation, there is high probablity that subsequent calls will also fail. So NotificationExecutor doesn't process notification for this plugin until the next iteration. Following message is shown in log:

```
Too many errors for plugin: <plugin-id> giving up for this  ↩
   session.
```

There was some records to sent in JMS queue, but unicore was unclean closed or freezed:

```
There was unsent records in the queue. You can use rus-usage ↩
   -logger-feeder to recover them!
```

10.3 rus-job-processor

Job processor uses in memory storage for created records. Capacity of records queue is 5000. If you stop unicore/X or unicore/X hangs while queue isn't empty then those records are lost. You can use rus-ussage-logger-feeder to parse logs and generate lost records. If connection with broker works fine, than records are sent immediately.

When queue size exceed 4000 records following warning is shown:

```
There is more then 4000 pending messsages to send. Please ↩
   check JMS broker connection.
```

When queue is full than following warning is shown:

```
Can't add message! Queue is full. Check your JMS connection.
```

Queue has limited capacity to avoid out of memory exceptions.

10.4 rus-bss-adapter

Refer to common problems. When broker connection is down, current record pointer doesn't advance. This approach gives you guarantee that any of record isn't lost. After connection is again established historical records are sent to broker.

10.4.1 Poison pill

When record line in BSS log isn't correctly parsed, then following error is shown:

```
Invalid record: <here goes record contents>
```

This line is simply ignored, since parsing proceduce is deterministic, and it could cause poison pill for bss-adapter.

10.5 rus-usage-logger-feeder

Refer to common problems.

10.6 rus-ucc-client

10.6.1 Rus-service is not installed or installed incorrectly.

Log:

```
[root@grid10 bin]# ./ucc accounting -s DEFAULT-SITE -m ↩
   grid10
2011-07-30 08:14:42,473 [main] ERROR HttpChannel  - Server ↩
   returned error code = 404 for URI :
        https://192.168.87.105:7777/services/RUS. Check ↩
           server logs for details
2011-07-30 08:14:42,475 [main] ERROR BaseUASClient  - Got ↩
   Error: class org.codehaus.xfire.
        XFireRuntimeException Could not invoke service.. ↩
           Nested exception is org.codehaus.xfire.fault.
        XFireFault: Server returned error code = 404 for URI ↩
            : https://192.168.87.105:7777/services/RUS.
        Check server logs for details
Error querying RUS Service at https://192.168.87.105:7777/ ↩
   services/RUS
The root error was: org.codehaus.xfire.XFireRuntimeException ↩
   : Server returned error code = 404 for
        URI : https://192.168.87.105:7777/services/RUS. ↩
           Check server logs for details
Re-run in verbose mode (-v) to see the full error stack ↩
   trace.
2011-07-30 08:14:42,477 [main] ERROR UCC  - Error querying ↩
   RUS Service at https://192.168.87.105:
        7777/services/RUS
org.codehaus.xfire.XFireRuntimeException: Could not invoke ↩
   service.. Nested exception is org.
        codehaus.xfire.fault.XFireFault: Server returned ↩
           error code = 404 for URI :
        https://192.168.87.105:7777/services/RUS. Check ↩
           server logs for details
        org.codehaus.xfire.fault.XFireFault: Server returned ↩
            error code = 404 for URI :
        https://192.168.87.105:7777/services/RUS. Check ↩
           server logs for details
        at org.codehaus.xfire.fault.XFireFault.createFault( ↩
           XFireFault.java:89)
        at org.codehaus.xfire.client.Invocation.invoke( ↩
           Invocation.java:83)
        at org.codehaus.xfire.client.Invocation.invoke( ↩
           Invocation.java:114)
        at org.codehaus.xfire.client.Client.invoke(Client. ↩
           java:336)
        at eu.unicore.security.xfireutil.client. ↩
           ReliableProxy.handleRequest(ReliableProxy.java ↩
           :122)
```

```
        at eu.unicore.security.xfireutil.client. ↩
           ReliableProxy.doInvoke(ReliableProxy.java:102)
        at eu.unicore.security.xfireutil.client. ↩
           ReliableProxy.invoke(ReliableProxy.java:69)
        at $Proxy13.extractRUSUsageRecordsByMachineName( ↩
           Unknown Source)
        at pl.edu.icm.unicore.accounting.ucc.Accounting. ↩
           getByMachineName(Accounting.java:274)
        at pl.edu.icm.unicore.accounting.ucc.Accounting. ↩
           process(Accounting.java:184)
        at de.fzj.unicore.ucc.UCC.main(UCC.java:179)
Caused by: org.codehaus.xfire.XFireRuntimeException: Server  ↩
   returned error code = 404 for
         URI : https://192.168.87.105:7777/services/RUS.  ↩
            Check server logs for details
        at org.codehaus.xfire.transport.http.HttpChannel. ↩
           sendViaClient(HttpChannel.java:130)
        at org.codehaus.xfire.transport.http.HttpChannel. ↩
           send(HttpChannel.java:48)
        at org.codehaus.xfire.handler.OutMessageSender. ↩
           invoke(OutMessageSender.java:26)
        at org.codehaus.xfire.handler.HandlerPipeline.invoke ↩
           (HandlerPipeline.java:131)
        at org.codehaus.xfire.client.Invocation.invoke( ↩
           Invocation.java:79)
        ... 9 more
```

Solution:

Verify your installation: Refer to point 10 in rus-service installation procedure.

10.7 rus-export-bat

Refer to common problems.

11 Changes

11.1 rus-ucc-plugin

11.1.1 2.0.0

- Updated to UCC 6.6.0
- Updated to new RUS-Service interface:

- It is possible to perform queries with multiple constraints.
- It is possible to perform summary report queries.
- New query constraints are available.

11.1.2 1.6.0

- Updated to UCC 6.5.0: short name of XML output opt. changed: $x \rightarrow$'X'.
- bugfix #3505249: removed option to query by submit host which is not implemented and won't be soon.

11.1.3 1.5.0

- feature #3389959: Re-sending plugin notifications (new command)
- improved -s handling: all Vsites can be used, not only TSFs
- updated to UCC 6.4.2, simplified installation
- better error messages
- more data from records is displayed

11.1.4 1.4.0

- removed getByXXX bug (spaces at the end of SOAP action names)
- changed long parameter name (duplicate --user)
- removed unnecessary libs, which reduced package size from 13 MB to 620 KB.
- clarified parameter names and output.

11.2 rus-service

11.2.1 2.0.0

- New database schema: unified identification of reporting and individual jobs records
- Added support for database rolling, i.e. automatic and configurable movement of old records to history tables.
- New query WS interface. The original RUS OGF Draft based interface fully dropped as unusable. New simple yet powerful interface implemented.

- Updated to USE 2.2.0 and basic security and configuration infrastructure of UNICORE 6.6, including SSL configuration for JMS.
- Fixed possible problems with DN comparison
- Added support for registering the service in UNICORE Registry
- Configuration is heavily updated. Among others only the master properties file is configured in UNICORE configuration, not a whole directory. Other configuration files are defined inside of it. SQL maps are not part of configuration directory anymore.

11.2.2 1.6.1

- bugfix: records with host which doesn't include description are handled properly now.

11.2.3 1.6.0

- use fixed ordering of notifications so better performance on heavy load is achieved
- added origin to export plugin condition context
- merging of records was revised and fixed in many cases
- creation time of merged record always have the earliest time
- bugfix: SubmitHost is properly merged now (previously it was ignored during merge so only the value from the first record was used).
- bugfix #3545929: DB connections limit is now correctly handled
- feature #3545930: Updated to MyBatis
- feature #3537691: Updated to USE 2.1, so rus-service can be installed on services from 6.5.0 release. This version is not compatible with older USE relases.
- Fixed a default location of the H2 database to be in data not in target/data
- JMS broker connection interval is increased when a subsequent connection failure is detected, so log files are less polluted.

11.2.4 1.5.0

- RUS interface class changed from pl.edu.icm.unicore.accounting.commons.ws.ResourceUsagePortType to pl.edu.icm.unicore.accounting.types.ws.ResourceUsagePortType
- rus-types extracted from rus-commons
- feature #3406826: don't wait when there are other records to process

- feature #3406829: Add more strict checking of records consistency
- sitename added to SpEL context
- feature #3389975: Configurable parameters for records processing
- feature #3389962: Better support for lost records from job-processor
- feature #3389973: Support for aggregate records
- feature #3389959: Re-sending plugin notifications

11.2.5 1.4.1

- fixed UVOS dependency issues

11.2.6 1.4.0

- introduces standalone version of rus-service: package with provides rus-serivce with Unicore container out-of-box.
- rus_extensions.xml is now provided out-of-box with two sample plugins commented out by default
- package is now shipped with XACML 2.0 policy
- XACML 1.0 policies was removed
- manual reorganization, cleanup, and enhancements
- spring rebase to 3.0.5
- updated status handling
- fixed getByMachineName

11.2.7 1.3.0

- support for Unicore 6.4
- created manual for accounting

11.2.8 1.2.0

- project refactoring

11.2.9 1.1.3

- replaced commons logging with log4j
- database dialect configuration merged to one file
- urs customization (removed duplicate namespaces to reduce size of UR's)

11.2.10 1.1.2

- rus_extensions.xml file watcher and automatic plugin config update
- merged database dialect configuration properties file
- commons-logging → log4j
- altered log levels (introduced TRACE log level)

11.2.11 1.1.1

- fixed broken MySQL support

11.3 rus-bss-adapter

11.3.1 2.0.0

- Minor changes to keep the codebase in line with other components:
- The JMS configuration mechanism is the same as in the other components.
- The overall configuration is based on the common UNICORE configuration framework.
- Added support for specifying normalization metrics

11.3.2 1.7.0

- Added support for SLURM.
- Backup of datapointer file is created before each modification. If original datapointer file is corrupted, the service won't start if backup exists.
- Added possibility to set a fixed sitename, for cases when no RUS job-processor is deployed for the BSS.

11.3.3 1.6.0

- Information about record origin is added
- fix: Torque main jobs of array job (with id xxxxx[]) in queued or started state are not reported anymore as were staying in this state forever. Only actual array member jobs are reported.
- Local identifiers of SGE array jobs are now reported as baseid[arrayIdx] instead of baseid-arrayIdx, so the same syntax as in Torque case is used.
- New feature: after 3 processing failures in a row, subsequent iterations are started after a longer, configurable delay.
- Proper parsing of lines with 3 tokens and with A, R, C and T codes.
- Updated debug logging to provide more useful information
- All records produced has a creation timestamp, so it is possible to better identify them, even before merging with other ones.

11.3.4 1.5.0

- Fixed #3390370: fatal errors are now logged to rus.log and startup.log
- Fixed bugs related to not closing file descriptors
- Collecting a maximum job completion time.

11.3.5 1.3.0

- added support for UNICORE 6.4

11.3.6 1.2.0

- new architecture: bss-adapter is feeding JMS queue instead of RUS-service

11.3.7 1.1.3

- log4j on-fly configuration
- log4j.properties changed to logging.properties

11.3.8 1.1.1

- added shutdown hook

11.4 rus-export-bat

11.4.1 2.0.0

- updated to the rus-service 2.0.0

11.4.2 1.5.1

- job id changed to free form string

11.4.3 1.5.0

- sitename is no longer required. When sitename is null, then sitename from ur is used.
- updated BAT schema to new version:
- added setting of submit_host attribute
- the job Id is integer now
- end time for not completed records is set to the maximum end time of the record

11.4.4 1.4.0

- rus_extensions relocated to conf/rus/ directory

11.4.5 1.3.0

- support for Unicore 6.4

11.4.6 1.2.0

- new architecture introduced

11.4.7 1.1.2

- bat converter: UNKNOWN value in BAT when infrastructure is empty in UR
- on-fly plugin swap enabled
- plugins have now close() method to release resources

11.5 rus-job-processor

11.5.1 2.0.0

- updated to the common libraries of the 2.0.0 release
- Added support for specifying normalization metrics

11.5.2 1.7.0

- Updated to U/X 6.6.0 API. No functional changes.

11.5.3 1.6.1

- Fixed numerous bugs in persistence configuration for fault tolerance.
- job-processor automatically tries to resend unsent records, also after container restart. Usage-logger-feeder is mostly not required anymore, except of the manual replying of jobs data.

11.5.4 1.6.0

- Records with information on approximate start, enqueue and end are sent. Data is in stored in separate fields to the precise BSS elements.
- Information about record origin is added
- bugfix #3390374: duplicated *queued* records are not sent anymore. Also some missing completed/failed records should be correctly sent now.
- updated to U/X release 6.5.0 API: selected VO is used.

11.5.5 1.5.0

- The module is included in Unicore/X distribution
- feature #3410407: Add information on the CE which submitted the job
- job-processor collects <resource desciption="sitename"> based on xnjs_legacy.xml property
- only one (or none) VO is reported for each job

11.5.6 1.4.1

- fixed UVOS dependency issues

11.5.7 1.4.0

- spring rebase to 3.0.5
- updated status handling, job-process now notify about job failure, abort or completition.

11.5.8 1.3.0

- support for unicore 6.4. Note: For unicore 6.3 use version 1.2.0.

11.5.9 1.2.0

- new architecture (rus-job-processor is derived from rus-service)

11.5.10 1.1.2

- rus_extensions.xml file watcher and automatic plugin config update
- merged database dialect configuration properties file
- commons-logging → log4j
- altered log levels (introduced TRACE log level)

11.5.11 1.1.1

- fixed broken MySQL support

11.6 rus-web-ui

11.6.1 2.0.0

-Renamed to rus-webui from ur-site

11.6.2 1.5.0

- Fixed #3406970: width of records' details view exceeds width of a browser
- Fixed #3390379: ur-site: filtering of VO-less jobs
- Fixed #3390371: Ur-site must be restarted after broker restart (just use failover:tcp:..,)
- support for container authN added
- site, which allows presentation of gathered data.
- better handling of job status

11.7 rus-usage-logger-feeder

11.7.1 1.4.0

- (with unicore.services.jobexecution.USAGE and USAGE)