EGEE

Apel User Guide

Document identifier:	**EGEE-JRA1-TEC**
Date:	**May 16, 2012**
Activity:	**JRA1: Middleware Engineering and Integration (UK Cluster)**
Document status:	**RELEASED**
Document link:	

Abstract: This document describes how to setup your site to be part of the Apel accounting system. It describes how to install and configure the Log Parser and Publisher components.

Document Change Log

Issue	Date	Comment	Author
5.0	03/12/10	Updated	Cristina del Cano Novales
4.0	18/05/10	Updated	Cristina del Cano Novales, Will Rogers
3.0	10/12/09	Updated	Cristina del Cano Novales
2.0	05/08/08	Updated	Cristina del Cano Novales
1.0	26/04/05	First Draft	Rob Byrom, Dave Kant

Document Change Record

Issue	Item	Reason for Change
5.0	6 Publisher	Updated with VO Filtering
2.0	2 Introduction	Updated with Savannah Patch 898
2.0	Figure 1	Updated
2.0	3.2 Installation	Removed reference to apel-db-update.py
2.0	3.2 Installation	Updated Bouncy Castle Provider path
2.0	4 Log Parser	Updated with Savannah Patch 898
2.0	Figure 2	Updated
2.0	Figure 3	Updated
2.0	4.3.4 CPUProcessor	Removed reference to default spec values
2.0	4.3.5 EventLogProcessor	Added SubmitHost
2.0	4.3.6 GKLogProcessor	Added new messages log file formats
2.0	GK/CE Configuration	Removed line from config file
2.0	6.3 Configuration	Updated with new variable Limit
2.0	10 Trouble Shooting	Updated

CONTENTS

1 DISCLAIMER 5

2 INTRODUCTION 6

2.1 WHERE ACTIVEMQ FITS IN 6

3 APEL CORE 7

3.1 DEPLOYMENT 7

3.2 INSTALLATION 7

4 LOG PARSER 8

4.1 DEPLOYMENT 8

4.1.1 SEPARATE GK/CE 8

4.1.2 COMBINED GK/CE 9

4.1.3 SAVANNAH PATCH 898 INSTALLED IN THE CE 9

4.1.4 MULTIPLE CES 9

4.2 INSTALLATION 9

4.2.1 LOG PARSER COMPONENTS 10

4.3 CONFIGURATION 10

4.3.1 PREAMBLE 11

4.3.2 DBPROCESSOR 11

4.3.3 DBDELETEPROCESSOR 11

4.3.4 CPUPROCESSOR 12

4.3.5 EVENTLOGPROCESSOR 12

4.3.6 GKLOGPROCESSOR 14

4.3.7 BLAHDLOGPROCESSOR 15

5 LOG PARSER EXAMPLES 16

5.1 SEPARATE GK/CE 16

5.1.1 GK CONFIGURATION 16

5.1.2 CE CONFIGURATION 16

5.2 COMBINED GK/CE 17

5.2.1 GK/CE CONFIGURATION 17

6 PUBLISHER 19

6.1 DEPLOYMENT 19

6.2 INSTALLATION 19

6.2.1 PUBLISHER COMPONENTS 19

6.3 CONFIGURATION 19

6.3.1 JOINPROCESSOR 20

7 PUBLISHER EXAMPLES **22**

7.1 GLITE-APEL SERVER CONFIGURATION 22

8 APEL ENCRYPTION SCHEME **25**

8.1 RSA KEY GENERATION 25

8.2 DETERMINATION OF KEY SIZE 25

8.3 PSEUDO RANDOM NUMBER GENERATION 25

8.4 ADDING RANDOM INFORMATION TO THE DN 26

8.5 HANDLING LONG DNS 27

8.6 DECRYPTING DATA 27

9 TROUBLE SHOOTING AND FAQ **28**

9.1 REPORTING PROBLEMS 28

1 DISCLAIMER

This guide provides an overview of the APEL accounting software.

APEL should be installed by following the generic gLite installation instructions[1].

If configuring manually, this guide describes where to install it and how to go about installing it.

[1]https://twiki.cern.ch/twiki/bin/view/LCG/GenericInstallGuide320

2 Introduction

The Apel software is composed of two components: The Log Parser and Publisher.

The Log Parser interprets log files to extract job information. Specifically, it processes the LCG gatekeeper logs, the system message logs and PBS, LSF, CONDOR or SGE event logs. When Savannah patch 898 is installed, the accounting log files are processed instead of the gatekeeper and messages files. Extracted data is then stored within a MySQL database. The Apel Log Parser also makes LDAP queries of the Computing Element to obtain the CPU performance figures for the worker node clusters.

The Publisher is used to generate accounting records derived from the parsed logging data. The accounting records are then published into ActiveMQ where they are then collected by a central accounting server which, aggregates records from all sites.

2.1 Where ActiveMQ fits in

The Apel software uses ActiveMQ as a transport mechanism for moving accounting records stored per site to a centralised database. This is achieved by using a producer to publish accounting records (performed by the Publisher) to an ActiveMQ broker. The records are read by a consumer into the centralised store (see Figure 1).

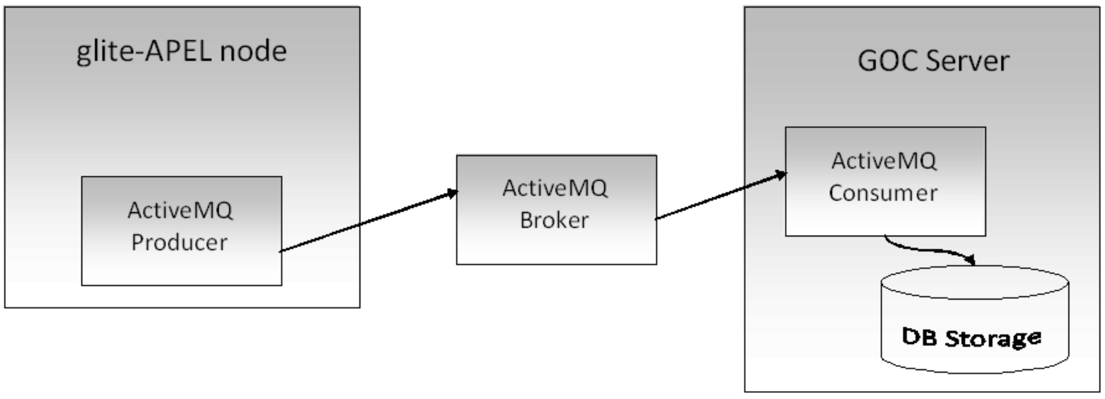

Figure 1: Accounting data is streamed from each site to a broker

For the purpose of this user guide, we assume the broker and consumer already exists (i.e. it's been configured by a sys admin located at the central accounting site).

3 APEL CORE

The Apel Core contains common functionality used by the Log Parser and Publisher components, it does not provide any direct functionality to the user.

3.1 DEPLOYMENT

Since the Log Parser and Publisher depend upon an Apel Core, this module must be deployed on all hosts running either component.

3.2 INSTALLATION

The Apel Core comes in rpm format available from the EGEE web site[2]. The package contains a number of files, the following lists those the sys admin should pay attention to:

- `/opt/glite/share/glite-apel-core/scripts/apel-db-config.py` this script is used to setup a MySQL database for a particular user. It will also install the schema used by both the Log Parser and Publisher. Make sure you run this script where your MySQL server is located.

On each host where the Apel Core is installed, make sure the following environment variable has been set:

```
export APEL_HOME=/opt/glite
```

If BC_PROVIDER is not set, APEL will assume it is located in either

```
/usr/share/java-ext/bouncycastle-jdk1.5/bcprov.jar
/usr/share/java/bcprov.jar
```

When installing the Apel Core, the apel-db-config.py script must be run on the same host as the MySQL server. Keep a note of the name of the database, user and password you use when running the script (you will need these when configuring the Log Parser and Publisher).

[2]http://glite.web.cern.ch/glite/packages/

4 Log Parser

The Log Parser is used to parse gate keeper, system message and event logs produced by a site running a batch processing system. When the CE has Savannah Patch 898 installed, the parser processes the accounting log files instead of gate keeper and message log files. Currently, the Log Parser comes in four different flavours: PBS, LSF, SGE and Condor. The Log Parser you choose will depend upon the batch processing system used by the site. However, the deployment and configuration are both equivalent.

4.1 Deployment

The Log Parser can be deployed in a variety of ways depending upon the setup of your site. An overview of some of the typical deployments are described as follows.

4.1.1 Separate GK/CE

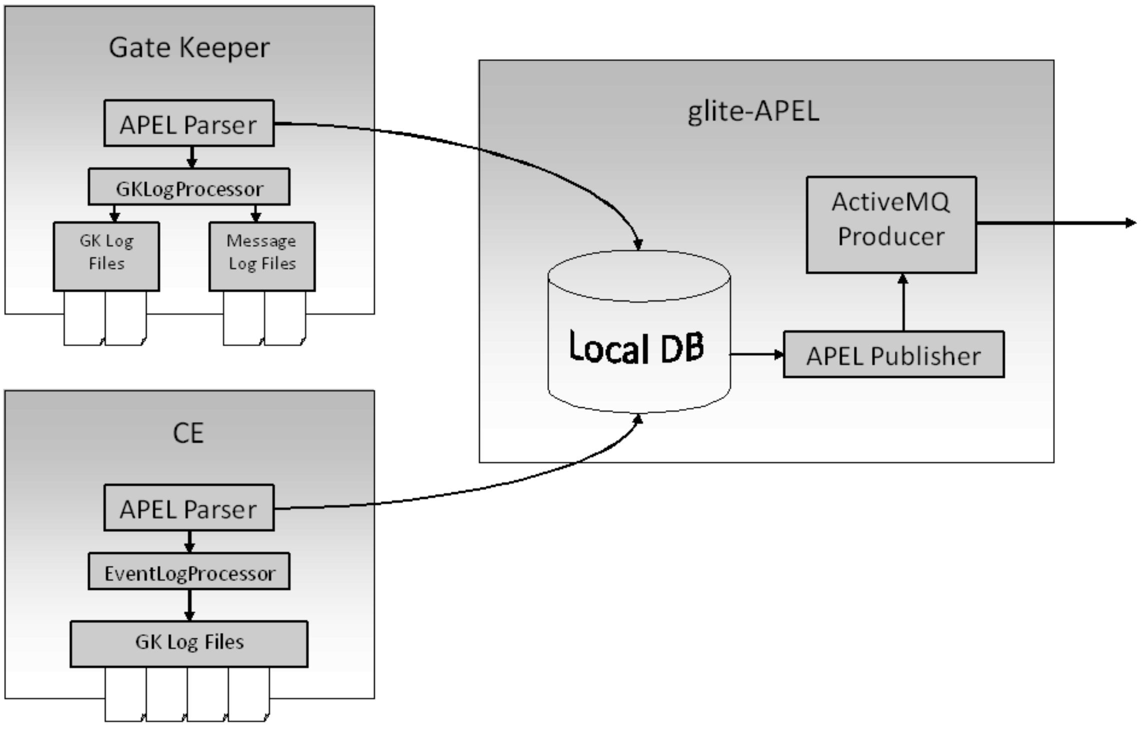

Figure 2: Shows Gate Keeper and CE on different hosts

See Fig 2. In this example, the gate keeper and CE are hosted on separate machines; both require an installation of the Log Parser. On the gate keeper, the Log Parser is configured to process gate keeper and message logs. Whereas the configuration on the CE is tailored to cater for event logs (conforming to the batch log format).

Fig 2 also shows the glite-APEL server. This is where the Publisher is normally installed (described in 6).

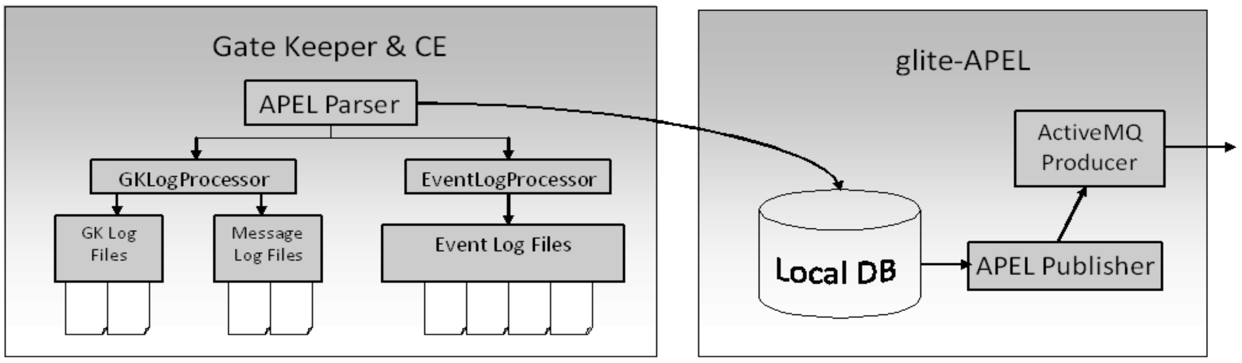

Figure 3: Shows Gate Keeper and CE combined on the same host

4.1.2 Combined GK/CE

See Fig 3. In this setup the Log Parser is configured to process both the gate keeper logs and event logs but on the same host.

4.1.3 Savannah Patch 898 Installed in the CE

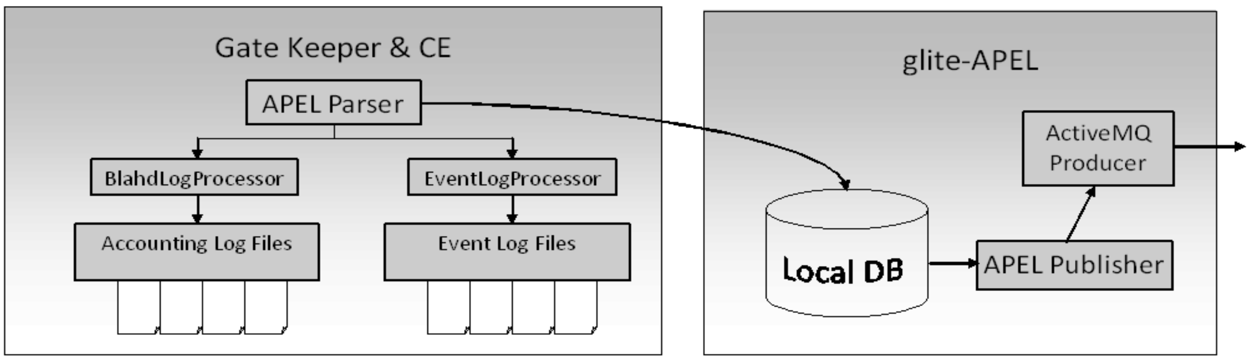

Figure 4: Shows CE with Savannah Patch 898 installed

See Fig 4. In this setup the CE has the Savannah Patch 898 installed. The GateKeeper and Messages log files are deprecated and the new Blah accounting log files are processed instead.

4.1.4 Multiple CEs

In the case of having multiple CEs in the site, the Log Parser must be installed in each CE.

4.2 Installation

The Log Parser comes with its own rpm package available from the EGEE web site[3]. It requires the Apel Core module to be installed (available from the same repository).

[3]http://glite.web.cern.ch/glite/packages/

4.2.1 Log Parser Components

The rpm installs the following components (using the PBS Log Parser as an example):

- `/opt/glite/bin/apel-pbs-log-parser` the script used to run the Log Parser.
- `/opt/glite/etc/glite-apel-pbs/parser-config.xml` contains an example config file used by the Log Parser.

The same files can be found when installing any of the LSF, SGE or condor Log Parser by substituting 'pbs' with 'lsf', CONDOR log parser substituting 'pbs' with 'condor' and SGE log parser substituting 'pbs' with 'sge'.

The Log Parser should be run daily. To schedule the Log Parser to run at 2:20 am every day, setup a cron entry as follows:

```
20 2 * * * /opt/glite/bin/apel-pbs-log-parser -f
/opt/glite/etc/glite-apel-pbs/config.xml > /var/log/apel.log 2>&1
```

In addition, it is a good idea to define a log rotation script, an example is show as follows:

```
/var/log/apel.log {
   copy
}
```

The log rotation script in this example is stored under `/etc/logrotate.d/apel`. Note, the cron and log rotate examples may need some tweaking depending upon the flavour of linux you use.

The sys admin should periodically backup data stored in the MySQL database used by the Log Parser. If tables grow large enough to cause disk space shortage, the sys admin is advised to back-up the database (using mysqldump) before purging the database contents.

4.3 Configuration

The Log Parser is setup using a configuration file encoded in an XML format. The configuration file is composed of a list of processors. Each processor carries out a unit of functionality.

When the Log Parser is started, the program will find any processors defined within the config file and will attempt to schedule each one consecutively. The list of configurable processors in order of execution are:

- `DBDeleteProcessor`
- `CPUProcessor`
- `EventLogProcessor`
- `GKLogProcessor`
- `BlahdLogProcessor`

An overview of the configuration format is detailed as follows.

4.3.1 PREAMBLE

The config file contains a preamble containing a number of fields that must be filled in by the user. The most important are the database credentials ie the database url, username and password. These credentials define the MySQL database to be used when storing parsed data derived from the gate keeper, system message and event log files.

An example of the preamble is shown as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<ApelConfiguration enableDebugLogging="yes">
    <DBURL>jdbc:mysql://grid-example.xy.ac.uk:3306/lcg</DBURL>
    <DBUsername>lcg</DBUsername>
    <DBPassword>no default</DBPassword>
    <SiteName>RAL-LCG2</SiteName>
</ApelConfiguration>
```

- `enableDebugLogging`: enables the user logging.
- `DBURL`: the URL that points to the MySQL database where the log data is to be recorded. Replace the hostname part with the hostname where your database is located.
- `DBUsername`/`DBPassword`: the username and password to be used to access the database on the MySQL database. Since a password is stored in this file you should setup suitable file access privileges.
- `SiteName`: is the name of the site producing the accounting data. **This must be set to the Sites' public Distinquished Name (DN) and must be consistent in all Apel config files used by the site**.

4.3.2 DBPROCESSOR

The `DBProcessor` is used to manage the APEL MySQL database schema. This processor examines the MySQL schema for compliance against the current version of APEL. Changes to the schema occur when new middleware releases allow for extra information to be captured in the accounting record.

Examples are any grid-level information that is mapped to local batch resources via the accounting log file (Savannah Patch 898). The UserFQAN derived from the users voms-proxy-init and taken from the LCMAPS mappings, the computing-element resource identifier and the global job identifier.

```
<DBProcessor inspectTables="yes"/>
```

- `inspectTable`: If enabled, the MySQL schema is interrogated. If APEL determines that an internal schema change is required, alterations will be performed. The time taken to complete this operation will depend on the size of the database. Once these changes have been done, subsequent schema interogations will not modify the table structure.

4.3.3 DBDELETEPROCESSOR

The `DBDeleteProcessor` is used to delete records from the accounting db. An example of the delete processor is show as follows:

```
<DBDeleteProcessor cleanAll="yes"/>
```

Special attention should be taken when using such a configuration. **As a general rule, separate the delete processor within its own config file to avoid any unwanted deletion while parsing new log files**.

- `CleanAll`: If enabled, all table contents in the accounting database will be purged. If disabled, only records that were successfully merged to produce accounting records are removed (this will only delete gate keeper, system message and event log file data stored in the database).

4.3.4 CPUPROCESSOR

If this processor is present, CPU performance information will be generated and inserted into the database to be used when publishing accounting records. Normally this information comes from an LDAP query to a GIIS server, specified by the `GIIS` element.

The default port used is 2170. If GIIS services are available on another port, the port number can be specified in the `GIIS` element using the following format: `host:port`.

An example using a GIIS host with default port 2170

```
<CPUProcessor>
    <!--
      Default: LDAP query to BDII port 2170 with "o=grid"
    //-->
    <GIIS>lcgce.example</GIIS>
</CPUProcessor>
```

An example using a GIIS host with a specified port:

```
<CPUProcessor>
    <!--
      LDAP query to GRIS port 2135 with "mds-vo-name=local,o=grid"
    //-->
    <GIIS>lcgce.example:2135</GIIS>
</CPUProcessor>
```

This processor should be run with the `EventLogProcessor` (see 4.3.5). **It is crucial that spec data is available when accounting records are generated by the Publisher**.

Note, if the GIIS is unavailable, the Log Parser will log the error and will terminate.

4.3.5 EVENTLOGPROCESSOR

The `EventLogProcessor` is used to parse event logs. The type of event logs parsed depend upon the log parser installed. This processor is usually run on the CE where the event logs are generated. An example config is shown as follows:

```
<EventLogProcessor>
    <SubmitHost>lcgce.example</SubmitHost>
    <Logs searchSubDirs="yes" reprocess="no">
```

```
        <Dir>/var/spool/pbs/server_priv/accounting</Dir>
        <ExtraFile>/home/toucan/pbslog00021123.gz</ExtraFile>
        <ExtraFile>/home/toucan/pbslog00021100.log</ExtraFile>
    </Logs>
    <Timezone>UTC</Timezone>
</EventLogProcessor>
```

- `SubmitHost`: should contain the hostname of the CE. This is the value that will be used as ExecutingCE in the accounting records.
- `Dir`: contains the directory where the event logs are to be found.
- `ExtraFile`: an optional parameter that allows additional filenames to be added to the list of event log files to be processed.
- `SearchSubDirs`: if enabled, all sub directories will be searched for event logs.
- `Reprocess`: when an event log file is processed, the log file name is stored in the database to prevent the file from be reprocessed in future. Enabling this option will force all event files to be re-parsed. Using this option is a safe way for re-building data from a batch of log files.
- `Timezone`: the event logs may be processed on a different machine from where they originated. For example, an admin user may want to run the Log Parser using event logs generated from a different site. Normally, the `EventLogProcessor` on each site will convert between UTC and local time of the processing host. However, this is not so useful where log files are being processed in a different timezone to that where the were generated.

  The behaviour can be modified using the following options:

  – `derived`: looks at the start and end time stamp of each job in the PBS event log (in UTC) to derive the timezone when the record was written.
  – `default`: uses the host's timezone.
  – `any other value`: for example 'GMT+1:30' indicates that 1 hour and 30 minutes has to be added to UTC to obtain the local time where the event log file was generated.

The name of the batch system log files must match the following format in order for the file to be parsed by APEL.

- `PBS`:

  ```
  YYYYMMDD
  YYYYMMDD.gz
  ```

- `SGE`:

  ```
  accounting
  accounting.\d{8}
  accounting.\d{8}.gz
  accounting.gz
  accounting-\d{8}
  accounting-\d{8}.gz
  ```

- `LSF`:

```
lsb.acct.\d{8}
lsb.acct.\d+
lsb.acct.\d{8}.gz
lsb.acct
```

- `CONDOR`:

```
history
history.gz
history.\d+
history.\d+.gz
```

4.3.6 GKLogProcessor

The `GKLogProcessor` is used to process to the gate keeper and system message logs generated on the Gate Keeper host. An example with an explanation of each field is shown as follows:

```
<GKLogProcessor>
    <!--
         <SubmitHost>lcgce.example</SubmitHost>
         No Longer used by this component
    -->

    <Logs searchSubDirs="yes" reprocess="no">
        <GKLogs>
            <Dir>/var/log</Dir>
        </GKLogs>
        <MessageLogs>
            <Dir>/var/log</Dir>
        </MessageLogs>
    </Logs>
</GKLogProcessor>
```

- `SubmitHost`: Not used by the GKLogProcessor.

  This is no longer used in the GKLogProcessor component. However, it may be used by the EventLogProcessor as Condor history and SunGridEngine batch logs do not provide the hostname of the submit node.

- `GKLogs`: contains the directory where the gate keeper log files will be found. gate keeper log file names contain a date suffix appended to `globus-gatekeeper` and may be of gzip format. File names examples are shown as follows:

```
globus-gatekeeper.log
globus-gatekeeper.log.20040125040202.0
globus-gatekeeper.log.20040201040203.0.gz
```

- `MessageLogs`: contains the directory where the system message log files are found. As with other files, these are recognised by having a particular filename format, examples of which are shown as follows:

```
    messages.1.gz
    messages.2.gz
    messages.200401201123904.gz
    messages.3.gz etc
```

- `SearchSubDirs`: if enabled all sub directories will be searched for gate keeper logs.
- `Reprocess`: Same as the attribute found in `EventLogProcessor`.

4.3.7 BLAHDLOGPROCESSOR

The `BlahdLogProcessor` is used to parse the Blah accounting log file on CEs where Savannah Patch 898 is installed. In this case the GateKeeper and the Messages log files do not need to be processed. An example config is shown as follows:

```
<BlahdLogProcessor>
    <!--
       The LRMS submit hostname.
    -->
    <SubmitHost>lcgce01.something.other</SubmitHost>

    <!--
         Please set <BlahdLogPrefix> to whatever is defined by the variable BLAHD_ACCOUNTING_I
         in /opt/glite/etc/blah.config

         Example: If BLAHD_ACCOUNTING_INFO_LOG is set to /var/log/glite/accounting/blahp.log

                  The output logs are /var/log/glite/accounting/blahp.log-YYYYMMDD and
                  <BlahdLogPrefix> is set to blahp.log
    -->
    <BlahdLogPrefix>blahp.log</BlahdLogPrefix>

    <Logs searchSubDirs="yes" reprocess="no">
        <Dir>/var/log/glite/accounting</Dir>
    </Logs>

</BlahdLogProcessor>
```

- `SubmitHost`: The batch server host name.
- `BlahdLogPrefix`: As explained above. The default log directory for accounting logs on the gLiteCE is /var/log/glite/accounting. All accounting files, which are rotated daily, have the name blahp.log followed by a the string ”YYYYMMDD”.
- `ExtraFile`: an optional parameter that allows additional filenames to be added to the list of event log files to be processed.
- `SearchSubDirs`: if enabled, all sub directories will be searched for event logs.
- `Reprocess`: when an event log file is processed, the log file name is stored in the database to prevent the file from be reprocessed in future. Enabling this option will force all event files to be re-parsed. Using this option is a safe way for re-building data from a batch of log files.

5 Log Parser Examples

5.1 Separate GK/CE

This example is based on the setup described in 4.1.1. The CE and Gate Keeper are situated on different machines each running the Log Parser. The configuration used for each Log Processor is described as follows:

- `GK`: will run the `GKLogProcessor`.
- `CE`: will run the `EventLogProcessor` and `CPUProcessor`

5.1.1 GK Configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<ApelConfiguration enableDebugLogging="yes">
    <DBURL>jdbc:mysql://grid-example.xy.ac.uk:3306/lcg</DBURL>
    <DBUsername>test</DBUsername>
    <DBPassword>info</DBPassword>
    <SiteName>gridka.de</SiteName>
    <GKLogProcessor>
        <Logs searchSubDirs="yes" reprocess="no">
            <GKLogs>
                <Dir>/var/data/gk</Dir>
            </GKLogs>
            <MessageLogs>
                <Dir>/var/message-data</Dir>
            </MessageLogs>
        </Logs>
    </GKLogProcessor>
</ApelConfiguration>
```

The Gate Keeper configuration only uses the `GKLogProcessor`. This processor will recursively search all directories below /var/data/gk and /var/message-data in-order to find any relevant log files. Since the 'reprocess' attribute is disabled, only new files will be processed.

Note, the DB credentials given are those used to access the DB located on the glite-APEL server (which contains an installation of the MySQL server).

5.1.2 CE Configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<ApelConfiguration enableDebugLogging="yes">
    <DBURL>jdbc:mysql://grid-example.xy.ac.uk:3306/lcg</DBURL>
    <DBUsername>test</DBUsername>
    <DBPassword>info</DBPassword>
    <SiteName>gridka.de</SiteName>
    <CPUProcessor>
        <GIIS>pbs-server2.gridka.de</GIIS>
    </CPUProcessor>
    <EventLogProcessor>
```

```
        <!--
             Add this line for processing Condor and Sun Grid Engine Batch Logs Only
             <SubmitHost>hostname</SubmitHost>
         //-->

        <Logs searchSubDirs="yes" reprocess="no">
            <Dir>/var/data/el</Dir>
        </Logs>

        <Timezone>UTC</Timezone>
    </EventLogProcessor>
</ApelConfiguration>
```

When the Log Parser is started, the `CPUProcessor` will be invoked first. It will try to query the GIIS (pbs-server2.gridka.de) in-order to retrieve a benchmark value of the site's CPU resources. Then all event logs found under /var/data/el will be processed (the search will include all sub directories).

5.2 COMBINED GK/CE

In this example, a site contains a GK and CE that, share the same host (based on the setup described in 4.1.2).

5.2.1 GK/CE CONFIGURATION

```
<?xml version="1.0" encoding="UTF-8"?>
<ApelConfiguration enableDebugLogging="yes">

    <DBURL>jdbc:mysql://grid-example.xy.ac.uk:3306/lcg</DBURL>
    <DBUsername>test</DBUsername>
    <DBPassword>info</DBPassword>
    <SiteName>RAL-LCG2</SiteName>

    <CPUProcessor>
        <GIIS>lcgce.example</GIIS>
    </CPUProcessor>

    <EventLogProcessor>
            <SubmitHost>hostname</SubmitHost>
        <Logs searchSubDirs="yes" reprocess="yes">
            <Dir>/var/spool/pbs/server_priv/accounting</Dir>
            <ExtraFile>/home/toucan/pbslog00021123.gz</ExtraFile>
            <ExtraFile>/home/toucan/pbslog00021100.log</ExtraFile>
        </Logs>
        <Timezone>UTC</Timezone>
    </EventLogProcessor>

    <GKLogProcessor>
        <Logs searchSubDirs="yes" reprocess="yes">
            <GKLogs>
                <Dir>/var/log</Dir>
            </GKLogs>
            <MessageLogs>
                <Dir>/var/log</Dir>
            </MessageLogs>
        </Logs>
    </GKLogProcessor>

</ApelConfiguration>
```

In this setup, the `CPUProcessor` is run first and is configured to query the GIIS server named lcgce.example. Then `EventLogProcessor` is will look for all log files located under the Dir path, it will also attempt to process the given extra files. The `GKLogProcessor` is the last to be scheduled and will process all GK logs. In this example all log the files found will be re-parsed.

Note, in both the GK and CE processors, the searchSubDirs option is enabled. This is the recommended option to ensure all files located under the given Dir are processed.

6 PUBLISHER

The Publisher is used to piece together accounting records derived from data parsed from gate keeper, system message and event log files. The generated data is stored locally within a MySQL database and is also published into ActiveMQ. The published data will then be stored by ActiveMQ onto the central accounting server.

6.1 DEPLOYMENT

In gLite 3.1, the publisher was deployed on the glite-MON server. However, this does not exist in gLite 3.2, and so APEL has it own server: gLite-APEL. MySQL and Java 1.6 should be installed on this node.

6.2 INSTALLATION

The Publisher comes with its own rpm package available from the EGEE web site[4]. It requires the APEL Core rpm to be installed (also available from the same repository).

Publisher machines are authenticated to the ActiveMQ broker using their X.509 certificates. This means that the glite-APEL node's DN needs to be properly registered in GOCDB.

6.2.1 PUBLISHER COMPONENTS

The rpm installs the following components:

- `/opt/glite/bin/apel-publisher` Script used to run the Publisher.
- `/opt/glite/etc/glite-apel-publisher/publisher-config.xml` Contains an example config file used by the Publisher.
- `/opt/glite/etc/glite-apel-publisher/jndi.properties` Contains an example config file specifying the ActiveMQ settings.

Similar to the Log Parser, the publisher should be scheduled to run on a daily basis. See notes described in 4.2.1.

6.3 CONFIGURATION

The Publisher is setup using a configuration file encoded in an XML format. Compared to the Log Parser, the configuration file is composed of only two processors: the `DBDeleteProcessor` and `JoinProcessor`

The configuration format uses the same header as that used by the Log Parser (see 4.3.1).

The `DBDeleteProcessor` is the same as that defined in 4.3.3.

The configuration file contains a variable Limit which will define the maximum number of records selected or published at any time by APEL publisher. This variable has been introduced to avoid OutOfMemory errors in the APEL publisher. Its value should be adapted to the amount of memory assigned to the Java VM. As a general rule:

- 512Mb - 150000 records

[4]http://glite.web.cern.ch/glite/packages/

- 1024Mb - 300000 records

Variables MaxBatchSize and ConsumerTimeout deal with ActiveMQ settings. It is recommended to leave the default values.

6.3.1 JOINPROCESSOR

The `JoinProcessor` is run on the glite-APEL server. It attempts to join data stored by the Log Parser to build accounting records. An example configuration is shown as follows:

```
<!--
publishGlobalUserName="no"  =>
         Do not send User DN information to GOC.
         UserDN information does not leave the site!
      publishGlobalUserName="yes" =>
         Send User DN information to GOC.
         The UserDN is encrypted using a 1024-bit-RSA key-pair.
      publicKeyLocation="/home/key/public.der"
         If set, this external public key will be used for encryption.
         Otherwise APEL key will be used
      disableJoin is optional and may be omitted.
//-->
<JoinProcessor publishGlobalUserName="no" disableJoin="no">
<Republish>all</Republish>
<!--
                VO Filtering:
                include="include"      : Only the VOs specified in VOList will be published
                include="exclude"      : All VOs except the ones specified in VOList will be
                include="all"          : All VOs will be published. VOList will be ignored.
                Enter one VO per <VO> tag

                VO Mapping:
                Map local unix groups to VOs if not using Blah Accounting Log files
                <VOGroup>localgroup:VOName</VOGroup>
        -->
        <VOInclude include="all">
                <!--VOList>
                        <VO>atlas</VO>
                        <VO>cms</VO>
                </VOList-->
                <!--VOMapping>
                        <VOGroup>atlasprod:atlas</VOGroup>
                        <VOGroup>cmsprod:cms</VOGroup>
                </VOMapping-->
        </VOInclude>
</JoinProcessor>
```

- `publishGlobalUserName`: if disabled, the Distinquished Name (DN) of the user in each accounting record is set to 'NULL' when published into ActiveMQ. Some sites may wish to suppress the

DN for reasons of *personal privacy*. By default, the DN will always be suppressed from publication. If enabled, the publisher will encrypt the UserDN information before publishing to GOC. The data will be decrypted by the GOC and access to this information restricted to the the individual User and VO administrator.

- `publicKeyLocation`: if set, APEL will use this public key to encrypt the UserDNs (only when publishGlobalUserName is enabled). If not set, APEL will use it's own internal public key.
- `disableJoin`: if enabled, APEL will not attempt to join data. If omitted or disabled, APEL will try to stitch accounting records together.
- `Republish`: Contains a list of options the user can choose when republishing accounting data. These are listed as follows:
  - `all`: setting the republish option to 'all' causes all accounting data in the database to be regenerated and published. **This option should never be set as a long term configuration. Doing so will impair the performance of the glite-APEL server especially if the volume of accounting records is large (order of tens of thousands)**.
  - `missing`: in some cases, the central accounting server may not receive the accounting records that were published in a previous join. Rather than having to republish the complete set, the `JoinProcessor` can send a subset of data begining from the last successful transfer. So when the `JoinProcessor` runs, a timestamp is recorded and the data is published as normal. When the `JoinProcessor` is next invoked, a count of all records that are greater than or equal to the timestamp is derived from the central accounting server. This is compared with the local database and if the accounting server contains fewer records, the `JoinProcessor` will republish all records begining from the timestamp. When the accounting server and local databases are in synch, the timestamp will then be updated as normal. **Note, this is the default behaviour**.
  - `nothing`: disables the republishing mode. Joining of records from the parser log tables will still be done.
  - `gap`: the gap publisher can be used to send accounting data to the central archiver in a defined period of time. The `recordStart` and `recordEnd` specifies the date when jobs started and finished at the site respectively. Unlike the `missing` option, the gap publisher does not alter timestamps, or attempt to stitch accounting data together. It just sends all accounting records that start and finish in the date range specified.
- `VOInclude`: this section specifies the mode of VO filtering and the list of VOs that will be included or excluded from publishing to the APEL central server. Please note: When using VO filtering the number of records joined will not match the number of records published to the server.
  - `include`: only the VOs specified in the VOList section will be published to the APEL server. **If no VOs are defined, the APEL publisher will not publish any data**.
  - `exclude`: all the VOs will be published except the ones defined in the VOList section. **If no VOs are specified in the VOList, all the VOs will be published**.
  - `all`: no filtering will be done. All the local data will be published to the APEL server.
- `VOMapping`: this section defines mappings between local unix groups and VOs. This is only needed when not using the blah accounting log files. Mappings are defined by the VOGroup tag (one line per mapping), which has the following syntax:

```
<VOGroup>localgroup:VO</VOGroup>
```

7 PUBLISHER EXAMPLES

The Publisher configuration remains unchanged irrespective of where the CE or GK is setup (see 4.1.1 and 4.1.2). This is because the Publisher is deployed on the glite-APEL server.

7.1 GLITE-APEL SERVER CONFIGURATION

An example Publisher config file used on the glite-APEL server is shown as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<ApelConfiguration enableDebugLogging="yes">
    <DBURL>jdbc:mysql://grid-example.xy.ac.uk:3306/lcg</DBURL>
    <DBUsername>gridka-mon</DBUsername>
    <DBPassword>mumbojumbo</DBPassword>
    <SiteName>gridka.de</SiteName>
    <!-- Number of records selected each time
 Modify this value if OutOfMemory error
 appears in the Publisher
 approx 150000 records per 512Mb memory -->
    <Limit>300000</Limit>
    <MaxInsertBatch>2000</MaxInsertBatch>
    <ConsumerTimeout>1800000</ConsumerTimeout>
    <!-- Examine APEL Schema //-->
    <DBProcessor inspectTables="yes"/>
    <!--
         publishGlobalUserName="no"  =>
                         Do not send User DN information to GOC.
                         UserDN information (encrypted or otherwise) does not leave the site.

         publishGlobalUserName="yes" =>
                         Send User DN information to GOC according to :-
                          Encrypted UserDN sent to GOC.
                          Unencrypted UserDNs are set to NULL before sending to GOC
    //-->

    <JoinProcessor publishGlobalUserName="no">
     <Republish>missing</Republish>
<!--
VO Filtering:
include="include" : Only the VOs specified in VOList will be published
include="exclude" : All VOs except the ones specified in VOList will be published
include="all" : All VOs will be published. VOList will be ignored.
  Enter one VO per <VO> tag

VO Mapping:
Map local unix groups to VOs if not using Blah Accounting Log files
<VOGroup>localgroup:VOName</VOGroup>
-->
<VOInclude include="include">
<VOList>
```

```
<VO>atlas</VO>
<VO>atlasprod</VO>
<VO>cms</VO>
</VOList>
<VOMapping>
<VOGroup>atlasprod:atlas</VOGroup>
<VOGroup>cmsprod:cms</VOGroup>
</VOMapping>
</VOInclude>
    </JoinProcessor>
</ApelConfiguration>
```

In this example, the `JoinProcessor` is setup to connect to the same MySQL database as the one used by the GK and CE. This is where the log data is held so the credentials of the DB are defined in this config file as well.

As for the VO filtering, only jobs by the atlas or cms VOs will be published to the central APEL server. This includes both records with primary FQAN for atlas/cms VO and local unix groups atlas and cms.

Records for local group atlasprod will be published with LCGUserVO atlas and local group cmsprod will be published as cms.

Please note that the mapping is done AFTER the filtering, so if you want to include records for local unix group atlasprod you need to define this in the VO list (it is not enough to just specify the mapping!).

An example gap Publisher to re-send all accounting data for last November:-

```
<?xml version="1.0" encoding="UTF-8"?>
<ApelConfiguration enableDebugLogging="yes">
<DBURL>jdbc:mysql://grid-example.xy.ac.uk:3306/lcg</DBURL>
<DBUsername>gridka-mon</DBUsername>
<DBPassword>mumbojumbo</DBPassword>
<SiteName>gridka.de</SiteName>
<!-- Number of records selected each time
         Modify this value if OutOfMemory error
appears in the Publisher
approx 150000 records per 512Mb memory -->
<Limit>300000</Limit>
<MaxInsertBatch>2000</MaxInsertBatch>
<ConsumerTimeout>1800000</ConsumerTimeout>
<!-- Examine APEL Schema //-->
<DBProcessor inspectTables="yes"/>
<!--
publishGlobalUserName="no"  =>
Do not send User DN information to GOC.
UserDN information (encrypted or otherwise) does not leave the site.

publishGlobalUserName="yes" =>
Send User DN information to GOC according to :-
Encrypted UserDN sent to GOC.
Unencrypted UserDNs are set to NULL before sending to GOC
//-->
```

```
<JoinProcessor publishGlobalUserName="no">
<Republish recordStart="2005-11-01" recordEnd="2005-11-30">gap</Republish>
<VOInclude include="exclude">
<VOList>
<VO>national_project</VO>
</VOList>
<!--VOMapping>
<VOGroup>atlasprod:atlas</VOGroup>
<VOGroup>cmsprod:cms</VOGroup>
</VOMapping-->
</VOInclude>
</JoinProcessor>
</ApelConfiguration>
```

The `JoinProcessor` will not create any new records from the parser log tables. However, it will publish any existing records in the specified gap (November 2005).

All the jobs will be published to the central server except for the ones by the VO national_project. No mapping of groups/VOs will be done as this section is commented.

8 APEL ENCRYPTION SCHEME

This section describes the method used by APEL to encrypt user DN information based on asymmetric private-public key encryption and a randomising function to reduce the likelihood of repeatable patterns arising in the ciphertext.

8.1 RSA KEY GENERATION

Using openSSL we generate a 1024 bit private key in PEM format.

```
openssl genrsa -des3 -out private.pem 1024
```

The private key is then used to generate the public key

```
openssl rsa -in private.key -pubout -out public.pem
```

The private key is converted from PEM to PKCS#8 format.

```
openssl pkcs8 -in private_key.pem -topk8 -v2 des3 -nocrypt -outform DER -out private_key.p8c
```

8.2 DETERMINATION OF KEY SIZE

The strength of encryption scheme, the size of input data string that can be encrypted, and the speed of the algorithm are important when encrypting data. 1024 bit keys are regarded as strong and a number of cash prizes on offer by RSA laboratories (key challenges) remain open. Unlike encryption with symmetric keys, which can be performed on plaintext of an arbitrary size, the maximum number of plaintext bytes that can be encrypted using a public key depends on the size of the key (and the type of padding). For example, when using a 1024-bit RSA key with PKCS #1 v.1.5 padding (one of the most common options), it is not possible to encrypt a string, which is longer than 117 bytes (that is 117 ASCII or 58 two-byte Unicode characters). Increasing the size of the RSA key to 2048 bits will allow you to encrypt up to 245 bytes of data, but longer RSA key are expensive: they take more time to generate and operate. Using long public keys (longer than 1024 bits) can seriously degrade application performance. Additionally, the cipher text string generated by a 2048 bit key - after base64 encoding has been performed - is greater than 300 characters, larger than the character limit allowed in a MySQL VARCHAR(255) field.

In EGEE, the longest DNs seen are approximately 117 bytes. After random data has been added to the DN (describe later) this increases to approximately 170 bytes. Thus, a 1024-bit key will not encrypt all DNs. To deal with these rare occurrences (less than 1 % of DNs seen in EGEE), an algorithm is applied to shorten the input string before encryption is applied. This is describe at the end of the section.

8.3 PSEUDO RANDOM NUMBER GENERATION

The SecureRandom class provided by the Bouncy Castle Crypto package is a Java implementation of cryptographic algorithms. This class provides a cryptographically strong pseudo-random number generator (PRNG). A cryptographically strong pseudo-random number minimally complies with the statistical random number generator tests specified in FIPS 140-2, Security Requirements for Cryptographic Modules, section 4.9.1. Additionally, SecureRandom must produce non-deterministic output and therefore it is required that the seed material be unpredictable and that output of SecureRandom be cryptographically strong sequences as described in RFC 1750: Randomness Recommendations for Security. When you create the class, either you pass it a seed, or it uses a self-generated seed. The seed is cryptographically hashed with the SHA-1 algorithm. Both the result and the internal state of the algorithm are added to the SHA hash to generate the next number.

If you don't pass in a seed, the seed is generated by the "seeder," which is itself an instance of SecureRandom. The seeder itself is seeded using internal thread-timing information that should be fairly hard to predict. The requisite timing information can take several seconds to gather, so the first time you create an instance of SecureRandom, this can be take up to 20 seconds.

The /dev/urandom device provides a reliable source of random output, however the output will not be generated from an equal amount of random input if insufficient input is available. Reads from the /dev/urandom device always return the quantity of output requested without blocking. If insufficient random input is available, alternate input will be processed by the random number generator to provide cryptographically secure output, the strength of which will reflect the strength of the algorithms used by the random number generator. The Output generated without random input is theoretically less secure than output generated from random input, so /dev/random should be used for applications for which a high level of confidence in the security of the output is required.

```
if (new File("/dev/urandom").exists())
  {
      System.out.println("Salting SecureRandom (SHA1PRNG) from /dev/urandom");
      byte[] salt = new byte[8192];
      new FileInputStream("/dev/urandom").read(salt);
      secureRandom.setSeed(salt);
      for (int t=0; t<salt.length; t++) salt[t] = 0;
      salt = null;
  }
```

8.4 Adding Random Information to the DN

Since many jobs can belong the same user DN, successive encryptions may lead to repeatable patterns in the ciphertext. To reduce the likelihood of this, two random numbers n1 and n2 are appended to the user DN string before encryption.

```
Original user DN:
   C=UK/O=eScience/OU=QueenMaryLondon/L=Physics/CN=dave kant
After adding random information
   956649486681*/C=UK/O=eScience/OU=QueenMaryLondon/L=Physics/CN=dave kant*30956649486681
```

Following recommendations from RSA laboratories and Bouncy Castle provider, we use an asymmetric block cipher with padding options: "RSA/ECB/PKCS1Padding" together with the GOC public key. The encryption algorithm also makes use of the SecureRandom function. The output cipher text is encoded into BASE64 format and can be written to a file or, in the case of APEL, sent as an encrypted stream via ActiveMQ to the database archiver. The encoded Cipher is 172 characters in length and can be easily accommodated as a database field with limit VARCHAR(255).

```
Output Cipher text (Base64 String Representation}
*APEL V.0.2*Z1R6Lg9CnTr2GZCeJt576Xpe/Dj3EXofwTTGppBzr5nmkt6Vmra7qVqZQXwOcrFi/
yYE8TYkSPl+RRg4kxdifCBBls1gZf4IB8VKf5Tfa+lOs2cRocd0jgrhOmVpt2h4P5lGYaJ+zrZYCl
MmoGjHs2KxOoZH3Tu+cpfUj5XH7RY=
```

As the encryption scheme may change in future implementations, e.g. with the support of robot certificates from CAs, a plaintext header is added to the cipher string to identify the method used to decrypt the data.

8.5 HANDLING LONG DNS

Known patterns in the UserDN e.g. */OU=personal certificate/* can be reduced to simply strings (/ou=pc/). Other techniques involve extraction of the email address from the string. Before any attempt is made to truncate the UserDN, the random numbers n1 and n2 are removed from the string and the strings size is checked before encryption occurs.

8.6 DECRYPTING DATA

Once the data has been sent via ActiveMQ to the central accounting repository, it remains in the ActiveMQ network in its encrypted form. To extract the UserDN, the data can be unencrypted using the header information together with the GOC private key. The random information can be easily removed. Once this is done, the unencrypted data MUST be removed from the ActiveMQ network and stored in a private database to protect the privacy of the accounting record. Access to the private database will be strictly prohibited and management of accounting data shall follow the guidelines specified by the GDB Accounting Policy document (in preparation).

```
Message decrypted with file private key:
 956649486681*/C=UK/O=eScience/OU=QueenMaryLondon/L=Physics/CN=dave kant*30956649486681
```

9 Trouble Shooting and FAQ

The FAQ for APEL can be found online at http://goc.grid.sinica.edu.tw/gocwiki/ApelFaq

9.1 Reporting Problems

Please report problems using the GGUS ticket service to either the *APEL Support Unit* or the *Accounting Policies Support Unit*. Tickets sent to these support units will be forwarded to apel-support@jiscmail.ac.uk. Try to include the following information:

- The siteName
- Description of the problem encountered
- Relevant parts of the APEL log file

REFERENCES