

EGEE

EGEE User's Guide

DGAS - GIANDUIA

Document identifier:	EGEE-JRA1-TEC-571271-GIANDUIA
Date:	November 4, 2005
Activity:	JRA1: Middleware Engineering and Integration
Lead Partner:	INFN
Document status:	DRAFT
Document link:	https://edms.cern.ch/document/571271/1

Abstract: Obtaining accurate resource usage records is essential for providing a reliable system for grid accounting, that can help in keeping track of the resource usage of single users as well as provide important statistical data for fair resource sharing among Virtual Organizations. This document describes the GIANDUIA service daemons required on a Computing Element (or a generic grid resource) in order to collect the usage records of the executed user jobs, as well as the clients and APIs that allow to publish them through the DGAS HLR service.

Delivery Slip

	Name	Partner	Date	Signature
From	A.Guarise, G. Patania, R.M.Piro	INFN Torino		
Reviewed by				
Approved by				

Document Change Log

Issue	Date	Comment	Author
Initial version	February 25,2005	-	A.Guarise, G. Patania, R.M.Piro
Updated version	August 5, 2005	updated Introduction, Installation and Configuration, added daemon start-up	A.Guarise, G. Patania, R.M.Piro
Updated version	September 23, 2005	minor improvements	A.Guarise, G. Patania, R.M.Piro
Updated version	November 4, 2005	additional configuration parameters; added handling of outOfBand jobs; added description of getAcctLogd and ceServiceClient; name change: now Distributed Grid Accounting System	A.Guarise, G. Patania, R.M.Piro

Document Change Record

Issue	Item	Reason for Change
-------	------	-------------------

Copyright ©Members of the EGEE Collaboration. 2005. See <http://eu-egEE.org/partners> for details on the copyright holders.

EGEE (“Enabling Grids for E-science in Europe”) is a project funded by the European Union. For more information on the project, its partners and contributors please see <http://www.eu-egEE.org>.

You are permitted to copy and distribute verbatim copies of this document containing this copyright notice, but modifying this document is not allowed. You are permitted to copy this document in whole or in part into other documents if you attach the following reference to the copied elements: “Copyright ©2005. Members of the EGEE Collaboration. <http://www.eu-egEE.org>”

The information contained in this document represents the views of EGEE as of the date they are published. EGEE does not guarantee that any information contained herein is error-free, or up to date.

EGEE MAKES NO WARRANTIES, EXPRESS, IMPLIED, OR STATUTORY, BY PUBLISHING THIS DOCUMENT.

CONTENTS

1	INTRODUCTION	4
1.1	GIANDUIA PURPOSE	4
1.2	SERVICE ARCHITECTURE	4
1.2.1	GIANDUIA	4
1.2.2	DGAS CE PUSHD	6
1.2.3	DGAS CESERVERD	6
1.2.4	DGAS GETACCTLOGD	6
1.2.5	THE HAD DAEMON	6
1.3	INTERACTIONS WITH OTHER SERVICES	7
1.3.1	DGAS USAGE RECORD ROUTING	7
1.3.2	DEFAULT USER HLR	8
2	QUICKSTART GUIDE	8
3	REFERENCE GUIDE	9
3.1	COMMANDLINE INTERFACES	9
3.1.1	ATM CLIENT	9
3.1.2	URWGMETERINGCLIENT	12
3.1.3	CESERVICECLIENT	13
3.1.4	LOGSERVICECLIENT	14
3.2	APPLICATION PROGRAM INTERFACES	14
3.2.1	ATM CLIENT API	14
3.2.2	URWGMETERINGCLIENT API	17
4	INSTALLATION	24
4.1	INSTALLATION OF GIANDUIA AND PUSHD ON THE COMPUTING ELEMENT	24
4.2	INSTALLATION OF THE CESERVERD CLIENT ON THE WORKER NODES	25
4.3	INSTALLATION OF THE GETACCTLOGD CLIENT ON THE LRMS HEAD NODE	25
5	CONFIGURATION	25
5.1	ATM CLIENT CONFIGURATION FILE	25
5.2	URWGMETERINGCLIENT CONFIGURATION FILE	26
5.3	GIANDUIA DAEMON CONFIGURATION FILE	26
5.4	CE PUSHD DAEMON CONFIGURATION FILE	28
5.5	GETACCTLOGD DAEMON CONFIGURATION FILE	31
6	RUNNING THE DGAS DAEMONS ON THE CE	32
7	KNOWN PROBLEMS AND CAVEATS	33

1 INTRODUCTION

The Distributed Grid Accounting System (DGAS), previously called DataGrid Accounting System, aims to be a full featured distributed Grid accounting toolkit. Since it is conceived and designed to be completely grid oriented, it is fully distributed without having a central repository of accounting information. It instead relies upon a network of independent accounting servers used to keep the accounting/transaction records of groups of Grid Users and Grid Resources.

DGAS can be used to account classic Computational Usage Records like CPU Time, memory usage and so on. It can also be used as an Economic Accounting system, treating information about the cost of the jobs executed by each Grid User on the single Grid Resources. This feature can be exploited for example by a Grid Service Provider that wants to charge its users for the provided service. As described in [3, 4, 5] the economic accounting can also be used to implement the so called *Economic Brokering* of the grid resources (selection of execution sites and services based on economic principles in order to improve the balancing of the workload).

1.1 GIANDUIA PURPOSE

Obtaining accurate resource usage records is essential for providing a reliable system for grid accounting. The GIANDUIA¹ service daemons are installed on a Computing Element (or a generic grid resource) in order to collect the usage records of the executed user jobs and send them to the DGAS HLR service [1] for accounting.

1.2 SERVICE ARCHITECTURE

Figure 1 shows a simplified deployment diagram of a Computing Element (CE) node, on which the software described in this document is installed, the User Home Location Register (HLR) node, that manages the account of the grid user that submitted a job to the CE, and a Resource HLR node, that manages the account of the CE that executed the user's job.

Two service daemons are responsible for sending accurate usage records from a CE node to the HLR service: Giandua for collecting the job Usage Records from the CE Local Resource Management System (LRMS)², and the CE Pushd for asynchronously forwarding them to the DGAS Resource HLR service³.

1.2.1 GIANDUIA

In order to collect the usage records, Giandua requires information about the existence of the grid jobs running on the CE. In order to inform Giandua the Job Wrapper creates a file (in a particular directory on the CE, see the following Section) containing the LRMS job name, the grid job id and the user HLR service address, that has been specified by the user upon job submission.

When a this job file is detected, the Giandua daemon looks for job metering information in the LRMS accounting log file. When the job has finished its execution the Usage Records for the job are retrieved from that log file and appended to the job file.

Thus for every grid job executed on the CE a file, containing the accounting Usage Records and the information needed to trace the job's grid identity, will be created in the so-called *dgasURBox*.

¹Giandua Is A Nice Distributed Usage-metering Infrastructure for Accounting

²Currently PBS and LSF LRMS are supported

³That is specified by the system manager with an appropriate configuration file

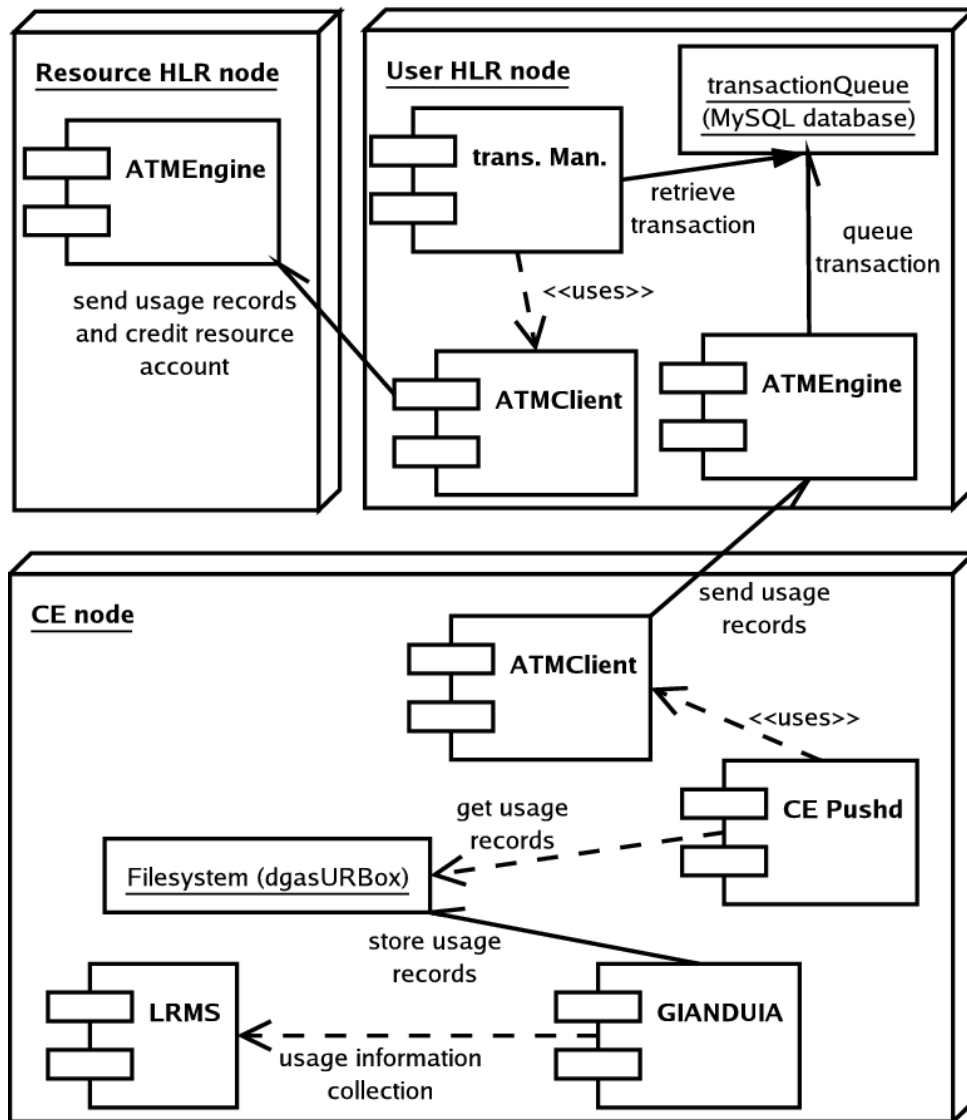


Figure 1: Simplified GIANDUIA Deployment Diagram. The diagram neglects many technical details.

1.2.2 DGAS CE PUSHD

This daemon uses the files created by Gianduia (or by another service that creates compatible files) and uses the information available in the file to initiate the transmissions of the usage records to the User HLR service, thus initiating the accounting procedures for the jobs.

The files created by Gianduia are treated in a queue and asynchronously processed. When a job's usage record is successfully sent to the User HLR, the corresponding file is removed from the queue and deleted such that it doesn't pollute the CE file system.

If a job's usage record can't be correctly transmitted, the process will be retried for a tunable amount of times, after which it will be marked as unprocessable. In this case the related information is not deleted such that it is still available to the CE site manager.

1.2.3 DGAS ceSERVERD

The *ceServerd* is a light weight daemon running together with Gianduia and collecting information transmitted from the Worker Node (WN) on which the job is actually running. The *ceServerd* is contacted by a equally light weight client that is run by the job's JobWrapper on the WN. Since the *ceServerd* is automatically handled by the Gianduia start-up script (see Section 6), it is not further described here.

1.2.4 DGAS GETACCTLOGD

The *getAcctLogd* is a light weight daemon that can be run together with Gianduia on the Computing Element. It is only necessary if the CE node and the LRMS head node do not coincide.

Some sites prefer to have the CE and the LRMS master on separate nodes. Since Usage records are composed from information coming from both CE and LRMS master log files, this daemon can be used to send the LRMS accounting logs, needed by Gianduia, to the CE.

The daemon runs on the Computing Element node where it listens on a TCP port. A client must be installed on the LRMS head node and should be periodically run via a `crond` script. The light weight client (see Section 3.1.4) simply reads a given text file (usually the accounting log file) and sends it to the remote peer (the CE) where it is stored so that the necessary accounting information is available to Gianduia.

The *getAcctLogd* is automatically handled by the Gianduia start-up script (see Section 6). It however requires to be properly configured (see Section 5.5).

1.2.5 THE HAD DAEMON

Since DGAS treats important information, it has to provide a high availability. The *High Availability Daemon (HAD)* is responsible for continuously monitoring the status of a service. In case of failure it restarts the daemon, thus avoiding long down periods due to service failures.

This is implemented with a perl script that takes the filename of a daemon startup script as its only command line argument. This, however, is automatically handled by the *Gianduia* and *CE Pushd* start-up scripts (described in Section 6) and thus the start-up of the HAD is not described here.

In order to be handled by HAD, the service startup script must understand the commands `status` and `restartMain`. The first one is the usual status command returning 0 if the daemon is alive and non-zero if it is stopped. The second one, `restartMain`, is a command that restarts all the processes managed by the script apart from the HAD daemon itself. The HAD daemon is managed by the service scripts standard commands `start` and `stop` as well, thus `restartMain` is required as a command that restarts

only the core services, but not the HAD; `restartMain` can then be used by the HAD to restart the core services, this avoids the HAD script to kill itself while attempting to restart the dead daemon.

1.3 INTERACTIONS WITH OTHER SERVICES

The only service that interacts with the DGAS-related services installed on the CE node, that is Giandua and the CE Pushd, is the DGAS HLR service (described in [1]). This interaction is unidirectional since the CE Pushd merely “pushes” the retrieved usage records of an executed job to the HLR server that manages the account of the user that submitted the job. As described later, this can be done using two different APIs/clients.

DGAS foresees two logical types of HLR servers.⁴ The *User HLR* and the *Resource HLR*. A User HLR stores information from a user's point of view and is the place where users can ask for accounting information concerning themselves and the jobs they have submitted. A Resource HLR stores information from a site manager's point of view and is the place where site managers, or resource owners, can ask for information concerning their resources.

The reason of this division is straightforward. For scalability reasons there will be many User HLRs on the grid, and different users will be registered with different HLRs. Hence it is necessary that all accounting information concerning a given user be forwarded (and stored) on the HLR that manages his account (“User” HLR).

On the other side a CE will get job submissions from different users that will be registered with many different HLRs. It is clear that a resource owner that needs an exhaustive view of the CE usage can't query all the possible User HLRs to get all the information needed (the same, of course, is valid for users that shouldn't need to query several Resource HLRs). Hence copies of the usage records for all the job executed on a CE must be present on another HLR that manages the CE's account (“Resource” HLR). Thus a resource owner needs to query only a single HLR to have the exhaustive accounting view needed, preserving in this way a reasonable scalability.

1.3.1 DGAS USAGE RECORD ROUTING

There are currently three scenarios for routing the information among the HLRs:

I) User-signed Dual HLR UR Routing (UDH-URR)

CE \Rightarrow (*user credentials*) \Rightarrow **User HLR** \Rightarrow (*User HLR credentials*) \Rightarrow **Resource HLR**

This is the default information routing. The usage record, collected on the CE, is signed with the user's credentials and sent over a secure channel to the User HLR. From there the UR is forwarded, signed with the User HLR's credentials to the Resource HLR. The process is a double commit. Since usage records are signed with the users' credentials, economic accounting can be asked to the system.

II) Resource-signed Dual HLR UR Routing (RDH-URR)

CE \Rightarrow (*CE credentials*) \Rightarrow **Resource HLR** \Rightarrow (*Resource HLR credentials*) \Rightarrow **User HLR**

This is an alternate route that can be used in two situations:

- The user's proxy certificate coming with the job is expired when the CE attempts to establish the communication with the User HLR (UDH-URR cannot be initiated);
- The CE's system administrator explicitly asks for this route.

⁴An HLR server can, however, manage both user and resource accounts if required.

In this scenario the job usage record is sent, signed with the CE credentials, over a secure channel to the Resource HLR. Then with the same double commit process, it is forwarded, signed with the Resource HLR credentials, to the User HLR. Since the UR is not signed with the user's credentials, economic accounting should be avoided.

III) Resource-signed Single HLR UR Routing (RSH-URR)

CE =(CE credentials)⇒ **Resource HLR**

In this scenario, the usage records are sent only to the Resource HLR. This is a requirement for some sites that do not want that their accounting information leaves their administrative domain. No information will be available to the User HLR concerning jobs executed on such sites. Economic accounting, through the exchange of virtual credits between the user's and the resource's account, is not possible.

1.3.2 DEFAULT USER HLR

For the scenarios I (UDH-URR) and II (RDH-URR), described in Section 1.3.1, it is necessary that the user specifies the contact string of the HLR (see Section 3.1) that manages the user's account (User HLR) in the JDL expression of the job (using the parameter `HLRLocation`⁵).

In some cases it can be useful to define a `defaultUserHLR` on the CE (in the configuration file of the CE *Pushd*, see Section 5.4), that is an HLR used by default when the `HLRLocation` parameter has not been specified by the user.

Clearly this default HLR must have accounts defined for all the users authorized to run on that CE. This is useful for testing purpose or on small grids if all user accounts are managed by a single HLR server.

2 QUICKSTART GUIDE

Since the collection and transmission of usage records is essential for providing a reliable system for grid accounting — that can help in keeping track of the resource usage of single users as well as provide important statistical data for fair resource sharing among Virtual Organizations —, it is essential to install and configure it properly, especially if the information it provides is used in an economic context in which grid users pay virtual credits for the resources they use. We therefore recommend to read the following Sections carefully (especially the Sections on installation and configuration), although we first make an example for the most common use case to give you an impression of the service furnished by Giandua and the CE *Pushd*.

Example 1: Submission of usage records using the `URWGMeteringClient`: Assume to have a usage record file, written by Giandua, in the `dgasURBox`. The usage records can be sent to the User HLR using the following command:

```
> $GLITE_LOCATION/libexec/glite_dgas_jobMeteringClientUrwg \
  -j "https://grid012g.cnaf.infn.it:6000/qaKyEoV3G144rmoyXeW6QA" \
  -B "hlr02.to.infn.it:56568:/C=IT/O=INFN/OU=Host/L=Torino/CN=hlr02.to.infn.it" \
  -P "pa02.to.infn.it:56567:/C=IT/O=INFN/OU=Host/L=Torino/CN=pa02.to.infn.it" \
  -r "grid002.to.infn.it:2119/jobmanager-pbs-long" \
  -u "/C=IT/O=INFN/OU=Personal Certificate/L=Torino/CN=Andrea Guarise/Email=\
  Andrea.Guarise@to.infn.it" \
  -s "hlr01.to.infn.it:56568:/C=IT/O=INFN/OU=Host/L=Torino/CN=hlr01.to.infn.it" \
  < /opt/glite/var/dgasURBox/<URfilename>
```

⁵In the future the parameter will be added as a VOMS extension to the user proxy, but currently the parameter has to be specified in the JDL

URfilename is the name of the file that contains a Usage Record in the GGF URWG XML format (and that is passed to the client as *stdin*). The parameter `-j` specifies the unique job ID associated to the usage record, `-B` the Resource HLR, `-P` the Price Authority (PA) responsible for setting the resource price, and `-r` the unique grid ID of the resource. The user that submitted the job is identified by the x509 certificate subject of parameter `-u` and the HLR server that manages to user's account (to which the usage record is sent) by the contact string of parameter `-s`.

3 REFERENCE GUIDE

3.1 COMMANDLINE INTERFACES

In this section we describe the command-line clients used by the CE Pushd to send the usage record of job to the User HLR⁶ server, where it will be archived and forwarded to the Resource HLR⁷. The usage record can then be accessed by both the user and the resource manager.⁸

Two HLR client programs are available: the first one, ATMClient — developed for the European Data-Grid Project —, is rather old and maintained for backward compatibility only; the second one, URWGMeteringClient, is new and implements the GGF Usage Record standard for exchanging usage records.

Note: The paths of the client programs are described using the environment variable `GLITE_LOCATION` that may be set during installation (the default value is `/opt/glite/`).

HLR and PA servers are usually specified by a *server contact string* that has the following form:

```
" [HOSTNAME] : [PORT] : [CERT_SUBJECT] "
```

where `[HOSTNAME]`, `[PORT]` and `[CERT_SUBJECT]` are the server's hostname, the port on which it is listening and the subject of its certificate respectively. For example:

```
-s "pa01.to.infn.it:56567:/C=IT/O=INFN/OU=DGAS_HLR/L=Torino\
/CN=pa01.to.infn.it/Email=andrea.guarise@to.infn.it"
```

3.1.1 ATM CLIENT

The ATM client can be used to to push already collected Usage Records to the User HLR service of the Distributed Grid Accounting System. Since it is deprecated we recommend to use the URWGMeteringClient that directly uses the usage record files as provided by Gianduia. The ATM client is maintained for maintained for backward compatibility only.

The command is:

```
$GLITE_LOCATION/libexec/glite_dgas_atmClient
```

having the following options:

```
> $GLITE_LOCATION/libexec/glite_dgas_atmClient <OPTIONS> [USAGE RECORD LIST]
DGAS ATM client
```

⁶The DGAS Home Location Register (HLR) server [1] that manages the account of the user that submitted the job.

⁷The HLR server that manages the account of the resource that executed the job.

⁸In a grid economy the usage record (together with the resource price) will also be used to compute the job cost, see [1] and [2].

Author: Andrea Guarise <andrea.guarise@to.infn.it>

Usage:

ATM_client <OPTIONS> [USAGE RECORD LIST]

Where options are:

-h	--help	Display this help and exit.
-j	--jobid <jobId>	Global job ID.
-t	--time <timestamp>	Submission time of the job.
-p	--paid <paID>	Contact string of the CE's Price authority.
-l	--localbankid <HLR contact>	Contact string of the (local) Resource HLR.
-C	--resgridid <ceID>	Global CE ID
-U	--usrcert <cert_subject>	User's X509 certificate subject
-r	--remhlrid <HLR contact>	Contact string of the remote User HLR.
-2	--toResourceHlr	Send record to resource HLR first
-3	--toResourceHlrOnly	Send record to resource HLR only

The HLR an PA contact strings have the form: "host:port:host_cert_subject".

The timestamp is specified in seconds since Jan. 1, 1970 0:00:00 GMT.

[USAGE RECORD LIST]:

```
CPU_TIME=<cputime> WALL_TIME=<walltime> PMEM=<physicalmem> VMEM=<virtualmem>
"QUEUE=<queuname>" "USER=<localuser>" "LRMSID=<lrmsid>" "PROCESSORS=<numproc>"
"URCREATION=<creationtime>" "group=<localgroup>" "jobName=<lrmsjobname>"
"start=<starttimestamp>" "end=<endtimestamp>" "ctime=<ctimestamp>"
"qtime=<qtimestampt>" "etime=<etimestampt>" "exitStatus=<exitstatus>"
"si2k=<specint>" "sf2k=<specfloat>" "tz=<numerictimezone>"
"fqan=<vomscertfqan>" "accountingProcedure=outOfBand"
```

Note: The mandatory usage metrics are: CPU_TIME, WALL_TIME, PMEM, VMEM, start, end, LRMSID and jobName; all other parameters are optional.

IMPORTANT: "accountingProcedure=outOfBand" should be specified when periodically processing LRMS logs instead of using DGAS Gianduia. This will cause the HLR server to effect additional controls to prevent from duplicate entries. This may also be used when the grid job ID is unknown (e.g. for jobs directly submitted to the CE without a UI that generates the grid job ID). In this case a unique job ID should be generated on the fly having the form "<hostname>:<lrmsID>:<timestamp>".

Example:

```
CPU_TIME=2 WALL_TIME=7 PMEM=1280KB VMEM=4472KB "QUEUE=short"
"USER=egEE003" "LRMSID=96128.lxb2077.cern.ch" "PROCESSORS=2"
"URCREATION=Mon Sep 5 17:39:45 2005" "group=gm" "jobName=blahjob_zul755"
"start=1123518309" "end=1123518315" "ctime=1123518299" "qtime=1123518299"
"etime=1123518299" "exitStatus=0" "si2k=400" "sf2k=380" "tz="+0200"
"fqan=/EGEE/Role=NULL/Capability=NULL"
```

The command line options correspond to the data members of the structures and the function described in Section 3.2.1. The parameters -p, -l and -r — for the CE's PA, the Resource HLR and the User HLR respectively — require contact strings (see Section 3.1), not only host names. The parameter -t is a timestamp that indicates the job submission time. The parameter -j is necessary to specify the unique

grid ID of the job to which the usage record has to be associated in the HLR service. The parameter `-U` specifies the certificate subject of the user that has to be accounted for the job. Finally, `-C` is the unique grid ID of the CE from which the the usage record is submitted, i.e. that executed the job.

The usage records are usually sent — using the user's certificate proxy — from the CE to the User HLR and from there to the Resource HLR (User-signed Dual HLR UR Routing). In case the proxy is expired they can also be sent to the Resource HLR first using the command line parameter `-2`. In this case the usage records are signed with the CE's host certificate (Resource-signed Dual HLR UR Routing). Alternatively they can be sent to the Resource HLR only (parameter `-3`), if no forwarding to the User HLR is desired (Resource-signed Single HLR UR Routing). For more detailed information on Usage Record Routing, see Section 1.3.1.

The `USAGE RECORD LIST`, that quantifies the amount of resource usage and specifies additional information on the job execution, is composed of the following key/value-pairs:⁹

- `CPU_TIME`: (MANDATORY) The consumed CPU time in seconds.
- `WALL_TIME`: (MANDATORY) The consumed wall clock time in seconds.
- `PMEM`: (MANDATORY) The consumed physical memory in kBytes.
- `VMEM`: (MANDATORY) The consumed virtual memory in kBytes.
- `QUEUE`: The name of the queue to which the job was assigned by the LRMS.
- `USER`: The local UNIX user with which the job was executed.
- `LRMSID`: (MANDATORY) The LRMS ID.
- `PROCESSORS`: The number of processors assigned to the job.
- `URCREATION`: The creation date and time of the usage record (for the format, see example below).
- `group`: The local UNIX group.
- `jobName`: (MANDATORY) The local LRMS job name/ID.
- `start`: (MANDATORY) The timestamp of the start of the job execution start.
- `end`: (MANDATORY) The timestamp of the end of the job execution.
- `ctime`: The timestamp of the time that the job was created by the LRMS (corresponding to the PBS `ctime`).
- `qtime`: The time that the job entered the current LRMS queue (corresponding to the PBS `qtime`; since the usage records are usually sent after job execution the value is in most cases equal to `etime`).
- `etime`: The timestamp of the time the job became eligible to run (i.e. entered an execution queue; corresponding to the PBS `etime`).
- `exitStatus`: The job's exit status.
- `si2k`: The worker node's `SpecInt2000` value.
- `sf2`: The worker node's `SpecFloat2000` value.

⁹Note that not all usage record parameters are mandatory.

- tz: The CE's time zone (numeric time modifier, see below for an example).
- fqan: The FQAN of the user's VOMS certificate.

Important note: When periodically processing LRMS logs instead of using DGAS Giandua (or additionally to using DGAS Giandua), you should specify "accountingProcedure=outOfBand" on the command line! This will cause the HLR server to effect additional controls to prevent from duplicate entries. This may also be useful if for some jobs the grid job ID is unknown (e.g. for jobs directly submitted to the CE without a UI that generates the grid job ID). In this case a unique job ID should be generated on the fly having the form "<hostname>:<lrmsID>:<timestamp>".

The following is an example for a complete USAGE RECORD LIST:

```
CPU_TIME=2 WALL_TIME=7 PMEM=1280kb VMEM=4472kb "QUEUE=short" \
"USER=egEE003" "LRMSID=96128.lxb2077.cern.ch" "PROCESSORS=2" \
"URCREATION=Mon Sep 5 17:39:45 2005" "group=gm" "jobName=blahjob_zu1755" \
"start=1123518309" "end=1123518315" "ctime=1123518299" "qtime=1123518299" \
"etime=1123518299" "exitStatus=0" "si2k=400" "sf2k=380" "tz="+0200" \
"fqan=/EGEE/Role=NULL/Capability=NULL"
```

The usage records of the example described also in Section 3.2.1 can be pushed on the DGAS HLR service with the following command:

```
> $GLITE_LOCATION/libexec/glite_dgas_atmClient \
-j "https://grid012g.cnaf.infn.it:6000/qaKyEoV3G144rmoyXeW6QA" \
-t 1103206983 \
-p "pa02.to.infn.it:56567:/C=IT/O=INFN/OU=Host/L=Torino/CN=pa02.to.infn.it" \
-l "hlr02.to.infn.it:56568:/C=IT/O=INFN/OU=Host/L=Torino/CN=hlr02.to.infn.it" \
-C "grid002.to.infn.it:2119/jobmanager-pbs-long" \
-U "/C=IT/O=INFN/OU=Personal Certificate/L=Torino/CN=Andrea Guarise/Email=\
Andrea.Guarise@to.infn.it" \
-r "hlr01.to.infn.it:56568:/C=IT/O=INFN/OU=Host/L=Torino/CN=hlr01.to.infn.it" \
CPU_TIME=100 WALL_TIME=1000 PMEM=10 VMEM=100
```

The parameter `-t` is optional too, since the corresponding timestamp is already available to the accounting system (due to the advance job authorization by the User Interface on behalf of the user, see Section "jobAuthEngine" in [1]).

The parameters `-p` and `-l` can also be specified in a configuration file (see in Section 5.1) such that all the jobs executed by the given CE share by default the same values for PA and Resource HLR contact strings. This should be the general case since the HLR account of a computing resource is usually managed by one particular HLR server as well as its price set by one particular PA server.¹⁰

3.1.2 URWGMETERINGCLIENT

This tool communicates Usage Records (already collected from the LRMS by Giandua). from the Grid Resource to the DGAS HLR service, implementing the GGF Usage Records Working Group (URWG) XML Schema [6, 7].¹¹

The command is:

¹⁰We propose to have one HLR and one PA server per Virtual Organization.

¹¹Please note: The work on the XML Schema for the communication of usage records is still ongoing. Hence, future changes are possible.

```

$GLITE_LOCATION/libexec/glite_dgas_urwgMeteringClient -h
glite_dgas_urwgMeteringClient
Author: Andrea Guarise <andrea.guarise@to.infn.it>
  
```

Usage:

```
cat <UR-XMLfile> | glite_dgas_urwgMeteringClient <OPTIONS>
```

Where options are:

```

-h --help                Display this help and exit.
-c --conf <confFile>    The ATM configuration file if not the
                        default (/opt/glite/etc/dgas_atmClient.conf)
-j --jobGridId <gJobID> Global grid job ID.
-P --resPaId <PA contact> Contact string of the CE's Price Authority.
-B --resBankId <HLR contact> Contact string of the Resource HLR server.
-r --resGridId <ceID>    Global CE ID.
-u --usrCertSubj <cert_subject> User's X509 certificate subject.
-s --usrBankId <HLR contact> Contact string of the User HLR server.
  
```

The <UR-XMLfile> is the file that contains the Usage Record in the XML format defined by the GGF UR-WG.

The HLR and PA contact strings have the form: "host:port:host_cert_subject".

where <UR-XMLfile> is a file that contains a Usage Record in the GGF URWG XML format and is passed to the client program as *stdin*. The parameter `-j` specifies the unique job ID associated to the usage record, `-B` the Resource HLR, `-P` the Price Authority (PA) responsible for setting the resource price, and `-r` the unique grid ID of the resource. The user that submitted the job is identified by the x509 certificate subject of parameter `-u` and the HLR server that manages to user's account (to which the usage record is sent) by the contact string of parameter `-s`.

The HLR and PA contact strings have to be formed as described in Section 3.1.

Note that this command has to be executed with the user certificate proxy available, because it is necessary to encrypt the Usage Record with the user key in order to guarantee confidentiality. Nonetheless the parameter `-u` is optional if the certificate used to run the command is that of the User who submitted the job, since the DN can be retrieved by the connection's security context.

As for the ATM client some of the parameters may be specified in the configuration file, described in Section 5.2.

For an example see Section 2.

3.1.3 CESERVICECLIENT

The service client is usually used by the JobWrapper to pass some grid-related information, that cannot be retrieved from the LRMS log, to Gianduia. It is not supposed to be used apart from that, but is still mentioned here for reasons of completeness. The command is:

```

$GLITE_LOCATION/libexec/glite_dgas_ceServiceClient -h
DGAS CE service client
Author: Andrea Guarise <andrea.guarise@to.infn.it>
  
```

Usage:

```
cat <userProxyFile> | glite_dgas_ceServiceClient <OPTIONS>
```

Where options are:

```

-h --help                Display this help and exit.
-s --server <ceServerd contact> Contact string of the ceServerd on the CE.
-L --lrmsJobId <lJobID>    LRMS Job ID of form: "lrmsType_jobid".
-G --gridJobId <gJobID>   Global grid job ID.
-H --hlrLocation <HLR contact> Contact string of the User HLR server.
-C --ceID <ceID>         Global CE ID.
  
```

The <userProxyFile> is the file that contains the user's proxy certificate.
 The HLR contact strings have the form: "host:port:host_cert_subject".
 The ceServerd contact string has the form: "host:port:"
 (usually: "<gatekeeper_host>:56569:").

3.1.4 LOGSERVICECLIENT

The `glite_dgas_logServiceClient` must be periodically executed (for instance in a cron script) on the LRMS head node if the LRMS head node is different from the CE node, see Section 1.2.4. The syntax of the command is:

```

$GLITE_LOCATION/libexec/glite_dgas_logServiceClient -h
DGAS log service client
Author: Andrea Guarise <andrea.guarise@to.infn.it>
  
```

Usage:

```
./glite_dgas_logServiceClient <OPTIONS>
```

Where OPTIONS are:

```

-h --help                Display this help and exit.
-s --server <ce hostname> Hostname of the server.
-p --port <port number>  Listening port of the remote Peer.
-f --file <file to send> File to send to the remote Peer.
-P --Pos <File used to store last position> File used to store temporary
                                     position info.
-c --conf <configuration file> Configuration file name
  
```

where the parameter `-s` is used to specify the CE hostname, `-p` the listening port of the `getAcctdLogd` daemon (see Section 1.2.4) on the CE, `-f` the log file to be sent to the CE and `-P` a file used by the client to store a pointer to the last file byte already sent to the server. This is necessary to keep track of already transmitted parts of log files since new information is usually appended at the end of the log file.

3.2 APPLICATION PROGRAM INTERFACES

In this Section we describe the C++ API of the two clients that can be used to forward a generic Usage Record to the HLR service.

3.2.1 ATM CLIENT API

Note: This API is deprecated, prefer using the `URWGMeteringClient` if available on your installation!

The API for the ATM client defines two data structures in the header file
`interface/glite/dgas/hlr-clients/atm/atmClient.h`.

```

struct ATM_job_data
{
    string dgJobId;
    int time;
    string res_acct_PA_id;
    string res_acct_bank_id;
    string user_CertSubject;
    string res_grid_id;
    string economicAccountingFlag;
};
  
```

```

struct ATM_usage_info
{
    int cpu_time;
    int wall_time;
    int mem;
    int vmem;
};
  
```

containing the following data members:

- `string dgJobId`: the unique Grid Job Id for the job.
- `int time`: a timestamp representing the moment in which the job has been registered to the accounting service.
- `string res_acct_PA_id`: the contact string of the Price Authority responsible for setting the price of the resource that executed the job.
- `string res_acct_bank_id`: the contact string of the HLR server that manages the account of the grid resource.
- `string user_CertSubject`: the X509 DN of the certificate of the user that submitted the job.
- `string res_grid_id`: the unique grid Id used to register the resource to the Resource HLR, usually it should be the CEId of the computing element.
- `string economicAccountingFlag`: `bool` (i.e. `true` or `false`) flag stating the user has to be charged for the execution of the job (required in a grid economy, but not necessary for accounting for statistical purpose).
- `int cpu_time`: seconds used by the CPU processing the user job.
- `int wall_time`: real duration in seconds of the user job.
- `int mem`: the maximum amount of physical memory in kilobytes used by the job.
- `int vmem`: the maximum amount of virtual memory in kilobytes used by the job.

The contact strings for PA and HLR servers have the form described in Section 3.1.

Currently the `user_CertSubject` in `struct ATM_job_data` is optional since the User X509 Certificate Subject is retrieved directly from the security context of the connection. It has been included for future enhancements, when the Usage Records will be sent to the Resource HLR in case the User HLR is not available or if the site manager needs so.

Instances of two structures described above are passed as parameters to the following function:

```

int ATM_client( string user_acct_bank_id,
               ATM_job_data &job_data,
               ATM_usage_info &usage_info,
               string *server_answer,
               string confFileName = GLITE_DGAS_DEF_CONF);

```

The function parameters are:

string user_acct_bank_id

The host address of the *User* HLR, that archives the Usage Records for accounting purpose and then forwards them to the *Resource* HLR.

ATM_job_data &job_data

A reference to the structure containing the information that identifies the user job (see above). **Note:** If the ATM_job_data data members res_acct_PA_id and res_acct_bank_id are empty strings, the values specified in the configuration file confFileName will be used. The value for economicAccountingFlag will always be taken from confFileName (if confFileName is a valid configuration file).¹²

ATM_usage_info &usage_info

A reference to the structure containing the Usage Records describing the job resource usage.

string *server_answer

A pointer to a string that, after the communication has finished, contains the XML answer from the server.

string confFileName

A string used to specify the name of the configuration file for the client. If it is NULL the default configuration file “/opt/glite/config/dgas_atmClient.conf”, defined in the header file interface/glite/dgas/hlr-clients/atm/atmClient.h, is used.

The function's return value is 0 if no error occurred, otherwise one of the error codes defined in the header file interface/glite/dgas/common/hlr/hlr_prot_errcode.h will be returned.

The following is an example for the usage of the API:

```

ATM_job_data job_data = {
    // dgJobId
    "https://grid012g.cnaf.infn.it:6000/qaKyEoV3G144rmoyXeW6QA",
    // time
    1103206983,
    // res_PA_url
    "pa02.to.infn.it:56567:/C=IT/O=INFN/OU=Host/L=Torino/CN=pa02.to.infn.it",
    // res_HLR_url
    "hlr02.to.infn.it:56568:/C=IT/O=INFN/OU=Host/L=Torino/CN=hlr02.to.infn.it",
    // usr_cert_subj
    "/C=IT/O=INFN/OU=Personal Certificate/L=Torino/CN=Andrea Guarise/Email=\
    Andrea.Guarise@to.infn.it",

```

¹²Economic accounting should be always active or always disabled. In case you really want to use it (or switch it off) for a single transaction for testing purpose you can “trick” the ATM_client function by specifying a non existing configuration file.

```

    // res_grid_id
    "grid002.to.infn.it:2119/jobmanager-pbs-long",
    //economicAccountingFlag
    true
};

ATM_usage_info usage_info = {
    100,      //cpu_time
    1000,    //wall_time
    10,      //mem
    100     //vmem
};

int res;
string output;
res = ATM_client("hlr01.to.infn.it:56568:/C=IT/O=INFN/OU=Host/L=Torino/CN=\
hlr02.to.infn.it",
                job_data,
                usage_info,
                &output
                );

```

3.2.2 URWGMETERINGCLIENT API

The class `urwg_UsageRecord` (in `interface/glite/dgas/common/hlr/dgas_urwg.h`) contains a set of data objects used to store Usage Record information. A constructor is defined for populating the data members using a string that contains the XML description of a Usage Record. Additionally, a method allows to produce the XML description of the Usage Record described by an instance of the class.

The data objects that store the usage records are based on the following basic data types:

```

typedef string dateTime;
typedef string description;
typedef string storageUnit;
typedef string phaseUnit;
typedef string metric;
typedef string type;
typedef string unit;
typedef string domainNameType;
typedef string duration;
typedef string ds_KeyInfo;

```

The single data objects that form the usage records are defined as:

```

struct intervallicVolume
{
    storageUnit storage;
    phaseUnit phase;
};

struct RecordIdentity

```

```

{
    dateTime createTime;
    string recordId;
};

struct JobIdentity
{
    string GlobalJobId; //minOccurs =0
    string LocalJobId; //minOccurs = 0
    vector<string> ProcessId; //minOccurs = 0 maxOccurs = unbounded
};

struct UserIdentity
{
    string LocalUserId; //minOccurs=0 maxOccurs=1
    ds_KeyInfo KeyInfo;//minOccurs=0 maxOccurs=1
};

struct JobName
{
    string Value;
    description Description;
};

struct Charge
{
    string Value;
    description Description;
    unit Unit;
    string Formula;
};

struct Status
{
    string Value;
    description Description;
};

struct WallDuration
{
    duration Value;
    description Description;
};

struct CpuDuration
{
    duration Value;
    description Description;
    string UsageType;
};

```

```
struct EndTime
{
    dateTime Value;
    description Description;
};

struct StartTime
{
    dateTime Value;
    description Description;
};

struct MachineName
{
    domainNameType Value;
    description Description;
};

struct Host
{
    domainNameType Value;
    description Description;
    bool primary;
};

struct SubmitHost
{
    domainNameType Value;
    description Description;
};

struct Queue
{
    string Value;
    description Description;
};

struct ProjectName
{
    string Value;
    description Description;
};

struct Network
{
    unsigned int Value;
    description Description;
    intervallicVolume IntervallicVolume;
    metric Metric;
};
```

```
};

struct Disk
{
    unsigned int Value;
    description Description;
    intervallicVolume IntervallicVolume;
    metric Metric;
    string Type;
};

struct Memory
{
    unsigned int Value;
    description Description;
    intervallicVolume IntervallicVolume;
    metric Metric;
    string Type;
};

struct Swap
{
    unsigned int Value;
    description Description;
    intervallicVolume IntervallicVolume;
    metric Metric;
    string Type;
};

struct NodeCount
{
    unsigned int Value;
    description Description;
    metric Metric;
};

struct Processors
{
    unsigned int Value;
    description Description;
    metric Metric;
    float ConsumptionRate;
};

struct TimeDuration
{
    duration Value;
    description Description;
    string Type;
};
```

```

struct TimeInstant
{
    dateTime Value;
    description Description;
    string Type;
};
  
```

```

struct ServiceLevel
{
    string Value;
    string Type;
};
  
```

The definitions of the data types and the data objects reflect the GGF URWG XML schema specifications [7]. Therefore refer to the schema for their meaning.

Additionally we need a structure to store DGAS specific information:

```

/*
 * Example of the XML output:
 * <Resource urwg:description="resAccountingPaId">paID</Resource>
 * <Resource urwg:description="resAccountingBankId">bankID</Resource>
 * */
struct DgasResourceExtension
{
    string resAccountingPaID;//PAID
    string resAccountingBankID;//resourceHLRID
    string userAccountingBankID;//userHLRID
    bool resRequiresEconomicAccounting; //default is false
    string resDgasCeId;//CeID
};
  
```

Now we can define a complete usage record:

```

class urwg_UsageRecord {
public:
    RecordIdentity recordIdentity;
    JobIdentity jobIdentity;
    UserIdentity userIdentity;
    JobName jobName;
    Charge charge;
    Status status;
    WallDuration wallDuration;
    CpuDuration cpuDuration;
    EndTime endTime;
    StartTime startTime;
    MachineName machineName;
    Host host;
    SubmitHost submitHost;
};
  
```

```

Queue queue;
ProjectName projectName;
vector<Network> network;
vector<Disk> disk;
vector<Memory> memory;
vector<Swap> swap;
vector<NodeCount> nodeCount;
vector<Processors> processors;
vector<TimeDuration> timeDuration;
vector<TimeInstant> timeInstant;
vector<ServiceLevel> serviceLevel;

DgasResourceExtension dgasResourceExtension;
urwg_UsageRecord();
urwg_UsageRecord(string &xml);
string xml();
}

```

As described above the constructor `urwg_UsageRecord(string &xml)` creates an instance of the class taking the necessary information from a usage record in XML format. The method `xml()` converts the information contained in an instance into an XML document.

As for the *ATM Client* (see Section 3.2.1) there is an API that directly implements the client used to establish a connection to the HLR server and send a usage record.

This job metering client, defined in `dgasJobMeteringClientUrwgImpl.h`, is implemented by the class `dgas::clients::dgasJobMeteringClientUrwgImpl` (that is derived from the more generic base class `dgas::clients::dgasClient`):

```

class dgasJobMeteringClientUrwgImpl : public dgasClient
{
public:
    string confFile;
    string xmlVersion;
    /**
     * Constructor
     * @param h hostname of the server
     * @param p portnumber of the server
     * @param c contact string of the server, optional.
     */
    dgasJobMeteringClientUrwgImpl(const std::string &h,
                                  int p,
                                  const std::string &c);
    /**
     * Destructor
     */
    ~dgasJobMeteringClientUrwgImpl();
    /**
     * Client init method, Wraps the communication with the server.
     * @param ur usage records for the job to account. This is the

```

```

    * GGF urwg implementation.
    */
int engage(dgas_common::urwg_UsageRecord &ur);

/**
 * Returns the answer received from the server
 */
string xmlServerAnswer(){return XmlServerAnswer;};

protected:

    string XmlServerAnswer;

    string composeXml(dgas_common::urwg_UsageRecord &ur);

    int parseXml(string& xml);
};

```

The class constructor is used to specify the remote HLR server to which to connect by specifying the host name (parameter *h*), the listening port (parameter *p*) and optionally the subject of the server host certificate (parameter *c*). If the host certificate subject is specified the connection will fail if the server presents itself with a different key.

The method `engage()` initiates the connection, sends the XML representation of the object of class `dgas_common::urwg_UsageRecord` passed by reference, and receives the XML answer back from the server. If no errors occurred the return value is zero.

The string containing the full XML answer can be obtained calling the method `xmlServerAnswer()`.

The following is a simple example for the usage of the client API:

```

// define the parameters for the constructor here
// ...

dgasJobMeteringClientUrwgImpl client(s,p,c);
urwg_UsageRecord u;

// ...
// populate the fields of the Usage Record here
// ...
cout << u.xml() << endl;

// eventually set a configuration file
// ...
if ( configFile_buff != "" )
{
    // As for ATM Client this API has its configuration file, too.
    client.configFile = configFile_buff;
}

// try to submit the resource usage record
// ...

```

```

try
{
    res = client.engage(u);
}
catch ( std::exception &e )
{
    cerr << "Exception caught: " << e.what() << endl;
    res = 1;
}

if ( res != 0 )
{
    cerr << "error: " << res << endl;
}
else
{
    cout << client.xmlServerAnswer() << endl;
}

```

4 INSTALLATION

The following is a brief description of the installation process for both the DGAS components required on the Computing Element and those required on the Worker Nodes.

4.1 INSTALLATION OF GIANDUIA AND PUSHD ON THE COMPUTING ELEMENT

The DGAS Metering System (Gianduia together with Pushd, ceServiced and getAcctLogd) has to be installed on the Computing Element (CE). Note that currently Gianduia and Pushd are part of the HLR clients package; in the future they will be delivered in an independent package.

The HLR clients, as all gLite components, should be installed on a machine running Scientific Linux CERN (currently version 3, see [8]).

The installation process can be divided into the following steps¹³:

1. Make sure that the following external globus packages (external packages for gLite on SLC3 can be found at [9]) are installed:

```

gpt
vdt_globus_essentials
vdt_globus_info_essentials

```

2. Make sure the CA certificates you need are installed (you may want to install all files beginning with "ca_*" from [9]).
3. Install the HLR clients (and the gLite components they depend on) using the RPMs from the latest available build (download them from [10]):

¹³Note that the version numbers mentioned in this document may differ from the current version numbers.

```

glite-wms-utils-exception
glite-wms-utils-jobid
glite-wms-utils-tls
glite-dgas-common
glite-dgas-hlr-clients
  
```

4.2 INSTALLATION OF THE CESEVERD CLIENT ON THE WORKER NODES

Additionally to the parts of the Metering System installed on the CE, DGAS requires a light weight client for the ceServerd (`ceServiceClient`, see Section 3.1.3), that sends important information to the Metering System on the CE, to be installed on each Worker Node (WN). Note that currently the ceServerd client is part of the HLR clients package; in the future it will be delivered in an independent package.

Since the ceServerd client is currently part of the HLR clients package, the installation process is the same as for Gianduia and Pushd (see Section 4.1).

4.3 INSTALLATION OF THE GETACCTLOGD CLIENT ON THE LRMS HEAD NODE

Additionally to the parts of the Metering System installed on the CE, DGAS may require a light weight client for the getAcctLogd (`logServiceClient`, see Section 3.1.4), that sends LRMS log information to the Metering System on the CE, to be installed on the LRMS head node. This is necessary ONLY if the the LRMS head node and the CE node do not coincide.

Since the getAcctLogd client is currently part of the HLR clients package, the installation process is the same as for Gianduia and Pushd (see Section 4.1).

5 CONFIGURATION

In this Section we describe the different configuration files for the clients and daemons described in the previous Sections. These configurations are necessary only on the CE node. Nothing has to be configured for the DGAS components on the Worker Nodes.

In the following, the keyword MANDATORY means that the parameter needs to be manually inserted in order for the system to work. The keyword OPTIONAL means that the system will work even if the parameter is omitted. The keyword DEFAULT means that the parameter doesn't need to be modified for the system to work. It can, however, be used to fine tune the system.

5.1 ATM CLIENT CONFIGURATION FILE

The ATM configuration file is usually named

```
$GLITE_LOCATION/etc/dgas_atmClient.conf
```

and has the following content:

```

# This is the configuration file for the ATMClient API, part of
# gLite DGAS, DataGrid Accounting System.
# Author: A.Guarise -- andrea.guarise@to.infn.it
# Author: R.M.Piro -- piro@to.infn.it
# Author: G.Patania -- patania@to.infn.it
  
```

```
#
#You should modify the following parameters according to your needs.
#

# next parameters specify the full contact string for the Resource
# PA and HLR.

#This configuration file is read by the DGAS glite_dgas_atmClient api
#And the info the following parameters are passed to the HLR of the user
#that has to be debited for a running job. These info are then used by that
#HLR to correctly process the economic transaction.

# resource PA: in the form host:port:X509_certSubject
res_acct_PA_id = "hostname:portnumber:X509CertSubject"
# resource HLR: in the form host:port:X509_certSubject
res_acct_bank_id = "hostname:portnumber:X509CertSubject"
# specify whether do we want economic accounting or not.
economicAccounting = "no"
```

The parameters of the ATM client configuration file are:

- **res_acct_PA_id (MANDATORY):** used to specify the contact string (i.e. the host address, port and certificate subject, see Section 3.2.1) of the DGAS PA server that is responsible for setting the price of the CE.
- **res_acct_bank_id (MANDATORY):** used to specify the contact string (see Section 3.2.1) of the DGAS HLR server that manages the CE's account.
- **economicAccounting (DEFAULT/MANDATORY):** used to specify whether economic accounting is required or not ("yes"/"no"), that is whether users should be charged virtual credits for resource usage or not. The default is "no".

5.2 URWGMETERINGCLIENT CONFIGURATION FILE

The URWGMeteringClient (described in Section 3.1.2) is optional and can be used instead of the ATM client (the CE Pushd uses the latter as a default). It currently uses the ATM configuration file (see Section 5.1).

5.3 GIANDUIA DAEMON CONFIGURATION FILE

The Giandua (also used for the `ceServerd` that is used with Giandua), configuration file is usually named

```
$GLITE_LOCATION/etc/dgas_giandua.conf
```

and has the following content:

```
# This is the configuration file for the 'giandua' UR metering daemon, part of
# gLite DGAS, DataGrid Accounting System.
```

```

# Author: A.Guarise -- andrea.guarise@to.infn.it
# Author: R.M.Piro -- piro@to.infn.it
# Author: G.Patania -- patania@to.infn.it

#
#You should modify the following parameters according to your needs.
#
#This configuration file is read by the DGAS 'gianduia' daemon

#This is the directory where blapd puts the files for the LRMS jobs with
# the information needed by the accounting service
chocolateBox = "/opt/glite/var/dgasRawBox/"
#This is the location of the directory where gianduia puts the files
#with the usage records. It must be the same as dgasURDir specified in
#the dgas_atmClient.conf file
gianduiottiBox = "/opt/glite/var/dgasURBox/"
#this is the location of the directory where garbage files are stored
#for post-mortem analysis
garbageCollector = "/opt/glite/var/garbageCollector/"
#the lock file for the daemon.
lockFileName = "/opt/glite/var/dgas_gianduia.lock"
#the log file name
logFileName = "/opt/glite/var/log/dgas_gianduia.log"
#chocolate Box parse interval in seconds
mainPollInterval = "60"
#garbage clean-up interval in seconds
queuePollInterval = "600"
#This is the location of the directory where PBS accounting logs are stored.
pbsAcctLogDir = "/var/spool/pbs/server_priv/accounting/"
#This is the location of the directory where LSF accounting logs are stored.
lsfAcctLogDir = "{path_lsf}/mnt/work/{cluster name}/logdir"

keyList = "GlueHostBenchmarkSF00,GlueHostBenchmarkSI00"
#ldifDefaultFiles = "file1,file2"
glueLdifFile = "/opt/glite/etc/glite-ce-ce-plugin/out.ldif"

dgasCeServerdLog = "/opt/glite/var/log/dgasCeServerd.log"
dgasCeServerdLock = "/opt/glite/var/dgasCeServerd.lock"
dgasCeServerdHadLock = "/opt/glite/var/dgasCeServerdHad.lock"

```

The parameters of the Gianduia client configuration file are:

- **chocolateBox (DEFAULT):** specifies the spool directory where the Gianduia daemon retrieves the usage record skeleton transferred from the JobWrapper of the job.
- **gianduiottiBox (DEFAULT):** specifies the directory where Gianduia puts the full usage record for the job once it is completed and the LRMS UR has been retrieved from the LRMS accounting log. **Important:** This directory **MUST** be the same as the one specified with the `dgasURDir` parameter of the CE Pushd configuration file (see Section 5.4).
- **garbageCollector (DEFAULT):** the location to which files are copied by Gianduia if severe errors occur during the collection of the accounting information.

- `lockFileName` (DEFAULT): the name of the Gianduia daemon process lock file.
- `logFileName` (DEFAULT): the name of the Gianduia daemon process log file.
- `mainPollInterval` (DEFAULT): the interval between the attempts to process the base usage records (building the full usage record from the skeleton and the information from the LRMS log). This value can be fine tuned in order to minimize the impact of the Gianduia daemon on the CE's workload.
- `queuePollInterval` (DEFAULT): the interval between two cleanups of the UR directory. During cleanup the daemon checks for garbage in the UR directory. This value can be fine tuned in order to minimize the impact of the Gianduia daemon on the CE's workload.
- `pbsAcctLogDir` (MANDATORY/DEFAULT for PBS/Torque based systems): the location of the directory in which PBS stores its accounting logs. It **MUST** be specified by the system administrator. However, the default value should be valid for most PBS installations.
- `lsfAcctLogDir` (MANDATORY/DEFAULT for LSF based systems): specifies where to find the LRMS accounting logs on LSF systems. The Gianduia daemon is able to find the value automatically on most installations. It is therefore necessary to specify it only on non standard installations of LSF.
- `keyList` (OPTIONAL/DEFAULT): a comma-separated list of parameters that Gianduia should retrieve from the `.ldif` files specified with the `ldifDefaultFiles` and `glueLdifFile` parameters. The key/value pairs will be appended to the usage record. The default keys "GlueHostBenchmarkSF00" and "GlueHostBenchmarkSI00", however, should not be deleted.
- `ldifDefaultFiles` (OPTIONAL): a comma-separated list of files to be searched for the keys specified with the `keyList` parameter.
- `glueLdifFile` (OPTIONAL): a single file to be searched for the keys specified with `keyList`. It overrides the contents of `ldifDefaultFiles`.
- `dgasCeServerdLog` (DEFAULT): the name of the `ceServerd` daemon process log file.
- `dgasCeServerdLock` (DEFAULT): the name of the `ceServerd` daemon process lock file.
- `dgasCeServerdHadLock` (DEFAULT): the name of the process lock file of the HAD daemon responsible for the `ceServerd`.

Important note: If the LRMS head node does not coincide with the CE node on which Gianduia is running and the LRMS log files are passed to the CE node using the `getAcctLogd`, the parameters `pbsAcctLogDir` and `lsfAcctLogDir` should point to the file(s) in which the `getAcctLogd` stores the information passed from the LRMS head node, see Section 5.5.

5.4 CE PUSHD DAEMON CONFIGURATION FILE

The configuration file for the *DGAS CE Pushd* daemon described in Section 1.2.2 is usually named

```
$GLITE_LOCATION/etc/dgas_ce_pushd.conf
```

and has the following content:

```

# This is the configuration file for the dgas_ce_pushd daemon, part of
# gLite DGAS, DataGrid Accounting System.
# Author: A.Guarise -- andrea.guarise@to.infn.it
# Author: R.M.Piro -- piro@to.infn.it
# Author: G.Patania -- patania@to.infn.it

#
#You should modify the following parameters according to your needs.
#

#The following parameters are needed by the glite_dgas_ce_pushd daemon.
#The daemon is recovers the UR of the job and asynchronously communicates
#them to the User HLR vie the glite_dgas_atmClient api.

dgasURDir = "/opt/glite/var/dgasURBox/"
dgasErrDir = "/opt/glite/var/dgasURBox/ERR/"
qDepth = "4"
qMult = "3"
lockFileName = "/opt/glite/var/dgas_ce_pushd.lock"
logFileName = "/opt/glite/var/log/dgas_ce_pushd.log"
mainPollInterval = "10"
queuePollInterval = "50"
#This is the User HLR where, by default, user records are sent.
#Change it according to your needs, or comment it out if you do not
#need a default userHLR
#defaultUserHLR = "grid003.mi.infn.it:56568:"

#if this flag is set to "yes" the UR will be sent to the resource HLR
#first. If it is commented or se to "no" they will be sent to the USer HLR first
#and then copied to the Resource One.
#forceLocalFirst = "yes"

#This specifies if UR shall be archived only in the resource HLR ("yes")
#or in the User HLR as well ("no")
forceLocalOnly = "no"

#This is the user used by the daemon to run the unpriviledged part
#of the script. it MUST not be 'root'. If it is empty, teh pool account
#of the user that runned the job will be used.
gridUser = "dgas"

```

The parameters of the CE Pushd configuration file are:

- **dgasURDir (DEFAULT):** specifies the spool directory where the daemon searches for the job usage records and user proxies.
- **dgasErrDir (DEFAULT):** specifies the spool directory where the daemon moves the usage record and user proxies that couldn't be processed after a given number of retries ($qDepth \times qMult$).
- **qDepth (DEFAULT):** specifies the depth of the daemon priority queue. Usage records traverse this queue before being moved in 'dgasErrDir'.

- `qMult` (DEFAULT): number of times the daemon tries to process the transmission of a usage record before lowering its priority in the queue.
- `lockFileName` (DEFAULT): lock file for the daemon process.
- `logFileName` (DEFAULT): log file for the daemon process.
- `mainPollInterval` (DEFAULT): time in seconds between two usage record processing cycles. This value can be fine tuned in order to minimize the impact of the CE Pushd on the CE's workload.
- `queuePollInterval` (DEFAULT): time in seconds after which the system processes lower priority usage records in the queue. This value can be fine tuned in order to minimize the impact of the CE Pushd on the CE's workload.
- `defaultUserHLR` (OPTIONAL): contact string (see Section 3.2.1) of an HLR that can be used as a default User HLR for users who's HLR server is not specified in the job's JDL or in the UI conf file. This feature must be used carefully.¹⁴
- `forceLocalFirst` (OPTIONAL): specifies an alternate routing for the usage record forwarding process between CE, User HLR and Resource HLR. If it is set to "yes", usage records are signed with the CE's host credentials and sent to the Site HLR (Resource HLR) first. If it is set to "no" usage records are signed with the user's credentials and sent to the User HLR first. See Section 1.3.1 for more information on DGAS usage record routing.
- `forceLocalOnly` (OPTIONAL/DEFAULT): if set to "yes" (default is "no"), it specifies that the daemon **MUST** sent usage records to the Resource/Site HLR **ONLY**. No copies of the usage records are sent to the User HLR. Usage record for jobs executed on this CE will be available to its Resource HLR **ONLY**. NO economic accounting is possible if this parameter is set to "yes". See Section 1.3.1 for more information on DGAS usage record routing.
- `gridUser` (MANDATORY/DEFAULT): If it is set to "" (empty string) the pool account that ran the user's job (e.g. atlas02) will be used to contact the User HLR with the user's credentials. If the string is not empty the specified UNIX account will be used instead (default is UNIX user "dgas"). There are two important considerations: A) If using a single account, such as "dgas", make sure the account has been properly created and has access to a shell (do **NOT** specify /sbin/nologin as the account's shell as is often done for pool accounts). B) Do **NOT** use "root" for this operation since in case of root the host certificate is taken for establishing the connection, not the user's credentials!

The directory specified by the `dgasURDir` parameter must contain two files for every job the site administrator wants to get accounted.

The first, having a name like `PBS_<pbsjobid>` or `LSF_<lsfjobid>`, contains the Usage Records as reported by Gianduia in the form:

```
HLR_LOCATION=hlr01.to.infn.it:56568:
GLITE_WMS_JOBID=<a grid jobId>
GLOBUS_RES_CONTACT=<global grid ID of the CE>
GlueHostBenchmarkSF00=<SpecFloat2000 value>
GlueHostBenchmarkSI00=<SpecInt2000 value>
currentTimeStamp=<UR creation time in seconds since 1970>
```

¹⁴The parameter has been added for testing purpose, but it may also be used for small Grid environments in which all users are registered with the same DGAS HLR server, see Section 1.3.2.

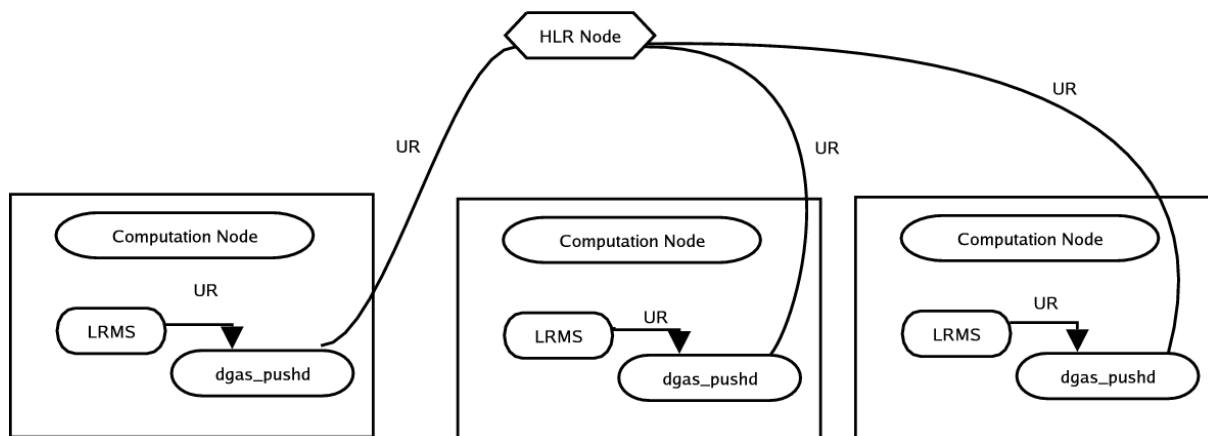


Figure 2: In this deployment example DGAS is used by a computing center to collect the Usage Records originating from its heterogeneous Computing Nodes.

```
timeZone=<timezone of CE, numeric, ex. '+0200'>
timeZoneText=<timezone of CE, textual, ex. 'CEST'>
ACCTLOG:<single line with the content of the LSF/PBS accounting log for the job>
```

where the first line is the contact string (see Section 3.2.1) of the User HLR to be contacted, the second line is the job ID and the last line is the LSF/PBS log of the job.

The second file has the same name plus the extension “.proxy”. It contains the proxy certificate required to secure the communication. Preferably it should be the user certificate, but it may also be the resource’s one, provided that the certificate is correctly mapped in the destination HLR specified by HLR_LOCATION value.

Ideally GLITE_WMS_JOBID should be the job ID provided by the WMS User Interface, but it might also be another unique jobId, for example “<CE hostname>:<jobs’ s LRMS ID>:<timestamp>”, if DGAS is used as a local Accounting Service instead of a grid-level one or when accounting jobs that were directly submitted to the CE without being assigned a global job ID by the User Interface (UI).

Figure 2 shows a simplified example for the deployment of a DGAS HLR server and three computing nodes with Gianduia/CE Pushd.

5.5 GETACCTLOGD DAEMON CONFIGURATION FILE

The configuration file for the DGAS *getAcctLogd* daemon described in Section 1.2.4 is usually named

```
$GLITE_LOCATION/etc/dgas_ce_getAcctLogd.conf
```

and has the following content:

```
listenerActive = "false"
aclFile = "/opt/glite/etc/getAcctLogd.acl"
listeningPort = "56565"
logFile = "/opt/glite/var/log/getAcctLogd.log"
lockFile = "/opt/glite/var/getAcctLogd.lock"
# if outputDir is specified instead of outputFile, the file name
# is received from the remote host.
```

```
outputFile = "/opt/glite/var/"
outputDir = "/opt/glite/var/"
```

The parameters of the getAcctLogd configuration file are:

- listenerActive (MANDATORY/DEFAULT): can be set to “true” or “false”. The listener is run only if it is set to “true” otherwise it is ignored, and the metering system takes the LRMS logs from the CE node assuming that it coincides with the LRMS head node.
- aclFile (MANDATORY/DEFAULT): Specifies the file for the host ACL. In this file the sys admin must specify the hosts that are allowed to send their logs to the CE (usually used to specify the LRMS master node hostname).
- listeningPort (DEFAULT): The port on which the daemon should listen.
- logFile (DEFAULT): The name of the file in which the listener logs its activities.
- lockFile (DEFAULT): The name of the listener’s lock file.
- outputFile (DEFAULT): The name of the file to which the daemon appends the contents of the file(s) sent by the authorized client(s).
- outputDir (DEFAULT): The directory in which the output file(s) is(are) written. If outputDir is specified, outputFile is ignored and the name of the output file will be the same as the one read by the client, and it will be put into the directory specified by outputDir. This is useful for instance when more than one file needs to be sent to the CE, which is for instance the PBS/Torque use case.

Important note: Since with this configuration the accounting log file (for LSF) or log dir (for PBS) are created by the listening daemon, the configuration file for the Gianduia daemon (see parameters pbsAcctLogDir and lsfAcctLogDir in Section 5.3) should be configured to point to the location of this file (or directory).

6 RUNNING THE DGAS DAEMONS ON THE CE

This Section briefly describes in examples how to run the DGAS daemons on the Computing Element. The ceServerd client on the Worker Node is launched by the job’s JobWrapper and does not need to be explicitly run by the administrator.

There are two startup scripts for the daemons:

```
/opt/glite/sbin/glite-wl-dgas-ce-gianduia
/opt/glite/sbin/glite-dgas-ce-pushd
```

Both scripts accept the standard options: start|stop|restart|status

To start-up the daemons simply use the following commands:

```
[root@lxb2077 sbin]# /opt/glite/sbin/glite-wl-dgas-ce-gianduia start
Starting dgas_gianduia.pl: [ OK ]
Starting /opt/glite/sbin/dgasCeServerd: [ OK ]
Starting ceServerd HAD... [ OK ]
```

```
[root@lxb2077 sbin]# /opt/glite/sbin/glite-dgas-ce-pushd start
Starting dgas-ce-pushd.pl: [ OK ]
```

IMPORTANT: These startup scripts are used not only to start-up the Gianduia and the Pushd daemons, but also to check and eventually prepare the environment, and moreover to start-up also the High Availability Daemon (HAD) responsible for checking that the key services are up & running and, in case of a service failure, to restart them (see Section 1.2.5).

7 KNOWN PROBLEMS AND CAVEATS

Most problems arise from bad configuration of the daemons or the clients used to submit the usage records to the HLR service. Make sure that your configurations respect the guidelines given in Section 5. Check your installation as well. Missing certificates (host certificates, as well as CA certificates), for example, will allow the CE Pushd to run, but the missing security context will not allow it to successfully contact the HLR servers.

REFERENCES

- [1] A. Guarise, G. Patania and R.M. Piro. "DGAS - Home Location Register. User's Guide". Technical Report EGEE-JRA1-TEC-571271-HLR. August 5, 2005.
- [2] A. Guarise, G. Patania and R.M. Piro. "DGAS - Price Authority. User's Guide". Technical Report EGEE-JRA1-TEC-571271-PA. August 5, 2005.
- [3] R.M. Piro, A. Guarise, and A. Werbrouck. "An Economy-based Accounting Infrastructure for the DataGrid". *Proceedings of the 4th International Workshop on Grid Computing (GRID2003)*, held in conjunction with the SC2003 Conference, Phoenix, Arizona, November 17, 2003.
- [4] R.M. Piro, A. Guarise, and A. Werbrouck. "Simulation of Price-sensitive Resource Brokering and the Hybrid Pricing Model with DGAS-Sim". *Proceedings of the 13th International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE 2004)*, track on Emerging Technologies for Next Generation Grid (ETNGRID 2004), Modena, Italy, June 14-16, 2004.
- [5] R.M. Piro, A. Guarise, and A. Werbrouck. "Price-sensitive Resource Brokering with the Hybrid Pricing Model and Widely Overlapping Price Domains". Accepted for publication in a special issue of *Concurrency and Computation: Practice and Experience* (Wiley Publishers).
- [6] GGF Usage Record Working Group.
<https://forge.gridforum.org/projects/ur-wg/>
<http://www.psc.edu/~lfm/Grid/UR-WG/>
- [7] R. Mach et al. "Usage Record – XML Format". Draft.
<http://www.psc.edu/~lfm/Grid/UR-WG/URWG-Schema.12.doc>
<http://www.psc.edu/~lfm/Grid/UR-WG/urwg-schema.11.xsd>
- [8] Scientific Linux CERN 3 (SLC3). <http://linux.web.cern.ch/linux/scientific3/>
- [9] External packages for gLite on SLC3. <http://glite.web.cern.ch/glite/packages/externals/bin/rhel30/RPMS/>

[10] Latest build of gLite packages. http://glite.web.cern.ch/glite/packages/#latest_builds