Leiden Grid Infrastructure

Dr. M.F. Somers

21 June 2022.
v 1.41

Table of Contents

Introduction:

"The Leiden Grid Infrastructure (LGI) is a framework that allows scientists to perform computational tasks and store data with a common interface without needing to know the details of the underlying systems. Moreover the infrastructure allows for a work and load balancing of the underlying systems; The calculation will run on a machine, being a resource within the grid, that can actually do the calculation. Data will be stored again on a resource within the grid in whatever format the resource can handle wherever there is place. Furthermore, with the LGI a scalable infrastructure is built that can grow whenever the demand increases for more resources."

The above stated general goal of the LGI can be implemented according to the infrastructural design, made clear in this document. The design is based on well-known and proven technologies.

Rationale:

Researchers and students of today cannot be expected to understand all the technicalities involved in using a computer or storage device efficiently; each system behaves differently in it's details and because the technologies involved in the computational field are growing in such a rapid rate, keeping up with that knowledge, together with keeping up with the scientific knowledge, is hard to do.

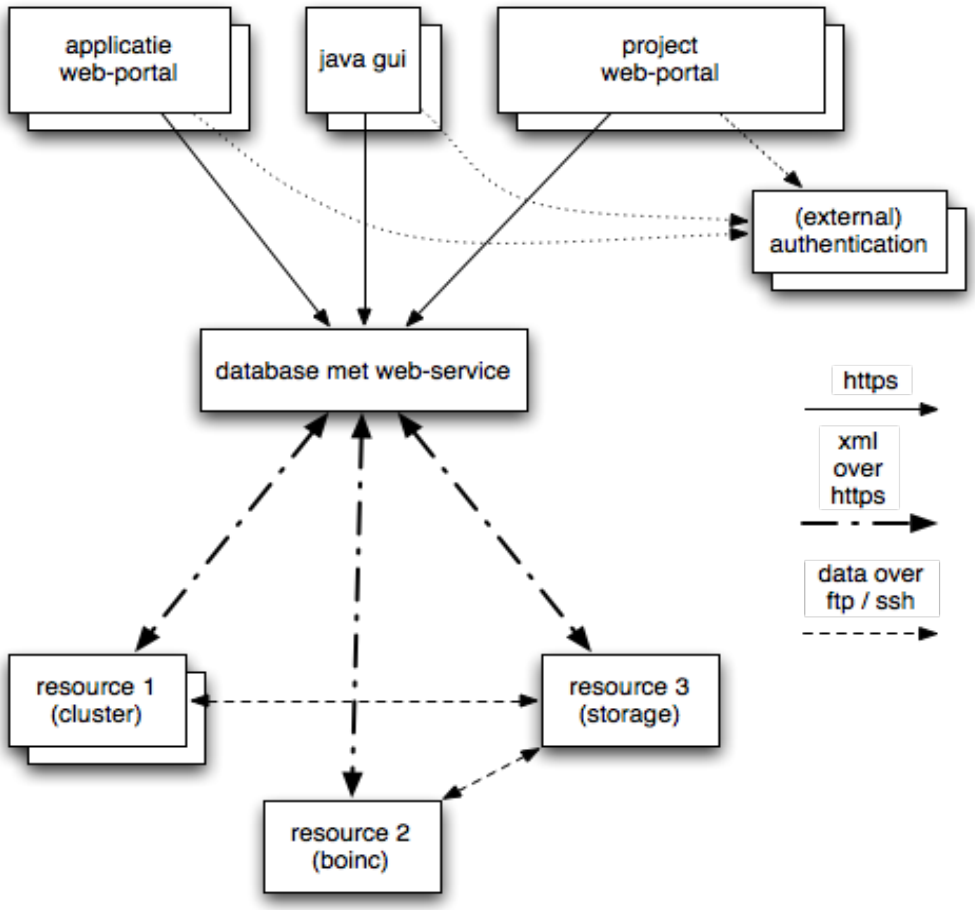
Past experiences on supercomputers and clusters has shown that on average such a system is made redundant within 6 years. Often, when continuing present work on newly made available systems, a lot of porting is involved and a detailed knowledge of the technicalities of the new hardware is needed. Keeping current software running in such a dynamic surrounding can be a rather time consuming job. Just think about the new technologies like having your programs run in parallel, either using OpenMP or MPI, or perhaps even both is the system allows for this... Also think about which compiler and with what setting to use for your new hardware...

A general scientist should not be bothered with all of this, especially if it is known that most of the software run on computers is usually compiled and set up by others, and all this is often restricted to only a few general applications. The same rationale applies to you own desktop system. You are probably using Windows, or perhaps Linux. What software was installed and why do you use it? Count the number of applications you use on a daily basis and see for how many of them you actually compiled the application from sources yourself!

Nowadays, computers can be used far more efficiently, both from a user, and from a management perspective, if the computers are part of a low-cost, perhaps application specific, but a very scalable grid infrastructure based on proven technologies.

General design:

The general LGI design is given by the following figure;



For each ***project***, any number of ***applications*** can be set up on any number of ***resources***. The ***application interface***, a common interface for an application within a project, communicates with any number of ***project servers***. One project server is designated to be the ***master project server***.

The project servers are nothing more than machines running a web-service to a general MariaDB/MySQL database with a known structure. The structure of the MariaDB/MySQL database will be detailed upon in this document: i.e. each project server contains an active list of all the resources within the LGI for that project and all the needed user management information. The project, within this framework, is nothing more than a specific MariaDB/MySQL database on each of the project servers. A single web-server machine can therefore automatically handle multiple projects independently.

The communication between an application interface and the project server is based on exchange of XML data over https using server- and client-certificates, all signed by the Leiden Grid Infrastructure Central Authority (LGI-CA). The LGI-CA is trusted by all clients running application interfaces, all project servers and all resources within the LGI.

The application interfaces queue calculation- or storage-jobs into any of the project server databases. This is done by requesting the project's master web-server to get a list of project server databases. This allows for a round-robin load balancing over these

project servers, if desired. An application interface can also monitor the completion of these jobs and depending on the details of the project, queue as much work as is allowed according to the users credentials. The user management for the project application can be implemented in the application interface itself but can also be set up on each of the project server database. The project servers only accept application interface clients with a valid LGI-CA singed certificate.

Application interfaces can be implemented in any programming language available as long as they communicate over https, using client- and server-certificates, all signed by the LGI-CA and adhere to the web-services offered by the project servers.

Resources, on the other end in this design, are nothing more than computers capable of doing calculations or storing data. A resource is identified with an unique resource name. In the project's master web-server an active resource table exists which couples the unique resource name to an user account on the resource (by using i.e. a 'somers@cartesius.surfsara.nl' type URL) and to it's LGI-CA signed certificate, which in turn contains the public part of the RSA key-pair of the resource, as usual. The unique name of the resource is also put into the client-certificate as the common name. More details on the use of certificates is given below.

A resource essentially runs a ***resource daemon*** in user-space and regularly polls the master and slave web-servers, perhaps even for different projects, to see if it is capable of performing any of the queued tasks for an application that was previously set up on that specific resource. The communication again makes use of XML over https, again with server- and client-certificates, all signed by the trusted LGI-CA. This allows for the project servers to identify requests through the verified client-certificates it receives.

The resource is regarded to be out of the administrating domain of the LGI; the owner and administrator of the resource can decide for which project's it will do what sort of work and for which users or groups of users for each of those project's applications. The resource daemon runs usually as a non-privileged user on the resource and has only a limited access to the resources available on the resource. This construct allows for, for example, regular user accounts on super-computers to be set up as a resource within the LGI for that specific project, user or group of users. More details on the resource daemon implementations are given in the following sections.

Within this design, storage and file transfers can be dealt with similar to computational tasks; storing a file is regarded to be a running 'file storage' job in the project server. It can be queued (only by a resource) for a specific resource, identified through the unique resource name within the project, or for any storage resource that is capable of storing the data in one way or the other. More details on how storage can be implemented in LGI are given below.

Database structure:

As is clear from the previous two sections, the project server, and with it the database, plays a central role in the LGI. A general and sound design of it's structure is therefore of out-most importance. In this part of the document some details on that matter are presented.

The project server contains a MariaDB/MySQL database for each project it might serve for. The name of the database is taken to be equal to the project name. In the project database the following tables are defined;

1) The table '***job_queue***';

field	***data type***
job_id	integer
state	string
application	string
owners	string
read_access	string
write_access	string
target_resources	string
input	blob
output	blob
job_specifics	string
lock_state	integer
state_time_stamp	integer
daemon_pulse	integer
priority	integer

This table shows the state for each job in the project server. It contains the 'job_id' assigned to the job, it's 'state', for what 'application' it is, for which 'target_resources', which groups or users are the 'owners' of the job, which groups or users have 'read_access' and which have 'write_access'. Only users and groups listed in the 'write_access' field can modify the job. The 'owners' field is used for accounting. If the 'target_resources' contains “any”, any capable resource can pick up the job. The 'owners', 'read_access' and 'write_access' fields are comma separated values and can contain the “any” keyword.

The 'job_specifics' may contain extra details on the specific job in XML format. It might contain extra scheduling information, specific to the application and the target_resources. The resource can decide to accept the job based on information present in that string. Currently the project server will insert tags here for a repository explained below. The 'input' and 'output' fields are general input and output fields. The interpretation of these fields depend on the application in question. They need not be used.

The 'state_time_stamp' is the time this job went into the state it currently is. The 'lock_state' field is used to signal a lock on this specific job. If the number is non-zero

positive, an entry in the 'running_locks' table should be present. Details on the locking mechanisms are given below.

The 'daemon_pulse' field is used to mark the last time a daemon checked the status of the job and the 'priority' field is used to deliver quality of service. Currently the 'priority' field is set equal to the submit time implementing a FIFO scheduling priority. A scheduler can change this 'priority' if desired.

2) The table '***running_locks***';

field	***data type***
lock_id	integer
job_id	integer
resource_id	integer
lock_time	integer
session_id	integer

This table will contain all the current running locks on the jobs of the 'job_queue' table for each session of resources. If a job in the 'job_queue' table is locked by a resource in a session, the 'lock_state' is a positive non-zero number in the 'job_queue' entry and an entry in this table is present. The UNIQUE 'job_id' indicates for which job in 'job_queue' this lock is, the 'resource_id' indicates, through the 'active_resources' table below, which resource has locked the job at what time, which is specified by 'lock_time' and the 'session_id' indicates what session of the resource has locked the job. The 'resource_id' field has been taken to be a signed integer to allow for interfaces locking jobs too: a lock with 'resource_id'=-1 and 'session_id'=0 is a lock from an interface rather than a resource. Details on the locking mechanisms are given below.

3) The table '***active_resources***';

field	***data type***
resource_id	integer
resource_name	string
resource_capabilities	string
client_certificate	blob
last_call_time	integer
project_server	integer
url	string

This table contains the currently signed-up resouces within the LGI for this specific project. The 'last_call_time' keeps track of the last time this resource has accessed this web-server. The 'project_server' defines what sort of project server the resource is; 0 = no project server, 1 = master project server , 2 = slave project server. The 'url' field specifies the location, on the web, and under what user account this resource, on the resource itself, is part of the grid. The 'client_certificate' contains a copy of the resource's certificate. Be sure to use correct naming convention for the resources:

'username@somewhere.org'. The 'resource_capabilities' XML field contains extra information the resource daemon has sent to the server during signing-up. The extra information is found from the daemon configuration.

4) The table '***running_sessions***';

field	***data type***
session_id	integer
resource_id	integer
session_time_stamp	integer

This table stores all the running sessions of all the resources. The 'session_id' uniquely identifies the session for the resource with id 'resource_id', The 'sessions_time_stamp' is a time value that is reset at each web request for which the resource needs a full session. If a session has timed-out for more than 30 minutes, the locks of that session are removed and the session itself is quited by the project server. Any service to the resource, for that specific session, is the denied.

5) The table '***users_allowed***';

field	***data type***
user_name	string
application	string
job_limit	integer

This table contains users who are granted access to the project server. If a specific user is not in this list, the record with the user_name equal to “any” can be used for the specific application, however, in that case the groups setting are first checked. The application entry can also be set to “any” to allow access to all applications within the project. If no such “any” 'user_name' is present in the list, the requesting user should be denied any service.

If the 'job_limit' value is negative, it specifies the maximum number of 'running' and 'queued' jobs this user is allowed to have for the application on this project server. If the number is positive it specifies the total number of jobs this user is allowed to have on this project server, whatever the state. If the number is zero, there is no limit.

User settings always overrule all group settings, unless the “any” user record was used and a specific group record is present in the 'groups_allowed' table.

6) The table '***groups_allowed***';

field	***data type***
group_name	string
application	string
job_limit	integer

This table contains the groups who are granted access to the project server. If a specific group is not in this list, the group should be denied any service for the specific application. This can be overruled by allowing for an “any” group. For application the “any” name can also be specified. However, in that case, the user tables are first consulted to check if there was no “any” user defined.

If the 'job_limit' value is negative, it specifies the maximum number of 'running' jobs this group is allowed to have for the application. If the number is positive it specifies the total number of jobs this group is allowed to have. If the number is zero, there is no limit.

Group settings are always overruled by user settings.

7) The table '***users_denied***';

field	***data type***
user_name	string
application	string

This table specifies which user is denied service for an application. This table overrules the 'users_allowed' and 'groups_allowed' tables. If the application is set to “any”, all service will be denied. The “any” 'user_name' is also allowed in this table.

8) The table '***groups_denied***';

field	***data type***
group_name	string
application	string

This table specifies which group is denied any service for an application. This table overrules the 'user_allowed' and 'groups_allowed' tables. If the application is set to “any”, all service will be denied. The “any” group can also be present in the table.

9) The table '***updates***';

field	***data type***
version	integer
servers	string
update_query	blob

This table contains all the updates this project server knows about. Each update is identified by a non-zero positive version number. If the update was intended for the specific server, listed in the comma separated 'servers' field, the 'update_query' has been applied to this servers project database. Only sequential updates are performed. Updates not intended for this server are still logged into this table. Because update version numbers are global to all project servers, updates should only be performed on the master project server even if they only apply to slave servers.

10) The table '***event_queue***';

field	***data type***
event_time_stamp	integer
event	string

This table is used to register and queue events for the project. Each time a job is added, changed state, or deleted, an event is scheduled. The table is also used to schedule a future event. The 'event_time_stamp' is the time at which the event is scheduled and 'event' is a string identifying the event itself. The default event 'schedule_cycle' will signal the scheduler a schedule cycle is needed. See below.

Resource interfacing with the project server:

In this part the API between the project server and a resource is detailed upon.

The web-interface to a project server will be implemented in php and by using XML data transfer over https, using client- server-certificates. Using these certificates, the resource can be identified through the 'common name' field in the certificate. All requests are made by passing data to the php scripts through the POST method. All responses from the project server will be wrapped into “<LGI> <response> </response> </LGI>” XML tags.

As of version 1.31 of LGI, two version number tags have been added designating the LGI project server version '<LGI_version> </LGI_version>' and the API version '<API_version> </API_version>'.

If an error condition has occurred during the request, a “<error> <number> 1 </number> <message> Back-off </message> </error>” type of response is generated. The non-zero positive numbers, enclosed in the “<number> </number>” tags, constitutes an error and has with it a corresponding message within the “<message> </message>” tags.

If a server is overloaded it should respond with the above 'back-off' error message. The resource should then try to look for the “<timeout> N </timeout>” tags, within the

“<error> </error>” message and wait for at least N seconds before trying again. It could also decide to try and request a different project server in the mean-time.

If a resource, not present in the web-servers 'active_resources' table does a request, an error is returned and service is denied. Also if the certificate of a resource does not match the one stored in the 'active_resources' table, service will be denied.

The following requests can be made to the web-server. They are listed under the 'resources' directory on the project server's main URL. For example “https://lgi.tc.lic.leidenuniv.nl/LGI/resources/resource_signup_resource.php”.

1) The request '***resource_signup_resource***';

This request is made by the resource prior to any use of the project's web-services. It allows the project server to setup a new session. The session number, which uniquely identifies this session, will be returned in the response and future requests should include that session id in their request when a full session is needed for that request. This request only needs the unique identifier of the resource, which is taken from the certificate's common name field, and the 'project' name, which is POSTed. It is possible to POST the 'capabilities' field to. If this XML field is posted, the 'resource_capabilities' record in the database is updated with this information.
The response, when an error occurred, on this request is an “<LGI> <response> </response> </LGI>” encapsulated error, as described above.

For a valid response, extra information can be placed by the web-server if needed, also encapsulated within the “<LGI> <response> </response> </LGI>” tags. Currently only the list of project's slave web-servers are exported by using the “<project_server number=x> url </project_server>” tags in which x is the logical number of the slave server. The total number of servers is also exported in the “<number_of_slave_servers> N </number_of_slave_servers>” tags.
An example response is given below:

```
<LGI>
        <CA_certificate> http://lgi.tc.lic.leidenuniv.nl/LGI/certificates/LGI-CA.crt </CA_certificate>
        <server_max_field_size> 65355 </server_max_field_size>
        <LGI_version> 1.34 </LGI_version>
        <API_version> 1.32 </API_version>
        <response>
                <resource> mark@testresource.somewere.nl </resource>
                <resource_url> </resource_url>
                <resource_capabilities> <hello_world> nothing </hello_world> </resource_capabilities>
                <project> testproject </project>
                <project_master_server> https://lgi.tc.lic.leidenuniv.nl/LGI </project_master_server>
                <this_project_server> https://lgislave.tc.lic.leidenuniv.nl/LGI </this_project_server>
                <number_of_slave_servers> 2 </number_of_slave_servers>
                <project_server number='1'> https://lgislave.tc.lic.leidenuniv.nl/LGI </project_server>
                <project_server number='2'> https://lgislave2.tc.lic.leidenuniv.nl/LGI </project_server>
                <session_id> 101 </session_id>
        </response>
</LGI>
```

2) The request '***resource_signoff_resource***';

This request is made by the resource to signal the end use of the project's web-services for the session identified through the posted 'session_id' field. It allows the project server to unlock any stale locks this resource might have on jobs in the project database and to kill the session it had with the resource. This request needs the unique identifier of the resource, which is taken from the certificate's common name field, the 'project' name, and the 'session_id' which should all be POSTed. The response generated to this request is exactly the same as described in the above 'resource_signup_resource' request.

3) The request '***resource_request_work***';

This request is made by a resource that seeks for work to be done. If an unregistered resource does the request, the request will be ignored and an error response will be send back, as was previously described. A valid session of the resource is required to be able to serve this request.
The common name field of the resource certificate is used to identify resource doing the request. The 'session_id', 'application' and the 'project' fields are set by a POST. The POSTing of the 'start' and 'limit' fields are optional in this request. If these fields are not POSTed, a default 'limit' of 10 will be used and if not POSTed, the 'start' is set to 0. If the 'limit' field was POSTed and is 0, a count of the number of jobs will be returned only and no jobs will be locked in that case.

The 'owners' field is optionally POSTed and allows for specifying a comma separated list of owners work is going to be allowed or denied for by the resource. Owner strings staring with the '!' are owners that will be denied work by the resource.

The web-server will query the 'job_queue' table, of the 'project' table in it's database for unlocked jobs with an offset of 'start' in the 'queued' state for the requesting resource. It will take the possibly POSTed 'owners' into account. It will generate locks for these rows on the fly as is described below and will respond with the XML data enclosed in the "<LGI> <response> </response> </LGI>" tags.

The response will have a "<number_of_jobs> N </number_of_jobs>" section that gives the amount of jobs returned in the response or the count of the number of 'queued' jobs available in the database. It will also have details on the requesting resource, project, the application of the request and the server's involved.

For each job in the response the "<job number=x> </job>" tags are used with x set equal to the logical number of the job in the response. Within these "<job> </job>" tags, the 'job_id', 'owners', the 'read_access', 'write_access', 'state_time_stamp' , 'the_job_specifics', and 'target_resources' are also specified. An example of a valid response is:

```
<LGI>
        <CA_certificate> https://lgi.tc.lic.leidenuniv.nl/LGI/certificates/LGI+CA.crt  </CA_certificate>
        <server_max_field_size> 65355 </server_max_field_size>
        <LGI_version> 1.34 </LGI_version>
        <API_version> 1.32 </API_version>
        <response>
                <start> 0 </start>
                <limit> 10 </limit>
                <resource> mark@testresource.somewere.nl </resource>
                <resource_url> </resource_url>
                <resource_capabilities> <hello_world> nothing </hello_world> </resource_capabilities>
                <project> testproject </project>
                <project_master_server> https://lgi.tc.lic.leidenuniv.nl/LGI </project_master_server>
                <this_project_server> https://lgislave.tc.lic.leidenuniv.nl/LGI </this_project_server>
                <session_id> 102 </session_id>
                <application> testapp </application>
                <number_of_jobs> 3 </number_of_jobs>

            <job number='1'>
                        <job_id> 140 </job_id>
                        <target_resources> testresource </target_resources>
                        <owners> mark </owners>
                        <read_access> mark, theor, sara, cyttron </read_access>
                        <write_access> mark, theor </write_access>
                        <state_time_stamp> 1259926661 </state_time_stamp>
                        <job_specifics> <nprocs> 2 </nprocs> <disk> 5G </disk> <mem> 64G </mem> <repository>
LGI@lgislave.tc.lic.leidenuniv.nl:/var/www/html/LGI/repository/JOB_000f61825ff3f59c736d737612e25e3d </repository>
<repository_url> https://lgislave.tc.lic.leidenuniv.nl/LGI/repository/JOB_000f61825ff3f59c736d737612e25e3d
</repository_url> </job_specifics>
                    </job>

            <job number='2'>
                        <job_id> 142 </job_id>
                        <target_resources> testresource </target_resources>
                        <owners> freek, cyttron </owners>
                        <read_access> cyttron </read_access>
                        <write_access> freek </write_access>
                        <state_time_stamp> 1259927661 </state_time_stamp>
                        <job_specifics> <nprocs> 1 </nprocs> <disk> 5G </disk> <mem> 64G </mem> <repository>
LGI@lgislave.tc.lic.leidenuniv.nl:/var/www/html/LGI/repository/JOB_010f61825ff3f59c736d737612e25e3d </repository>
<repository_url> https://lgislave.tc.lic.leidenuniv.nl/LGI/repository/JOB_010f61825ff3f59c736d737612e25e3d
</repository_url> </job_specifics>
                </job>

            <job number='3'>
```

```
                                <job_id> 147 </job_id>
                                <target_resources> any </target_resources>
                                <owners> sjoerd, cyttron </owners>
                                <read_access> any </read_access>
                                <write_access> cyttron </write_access>
                                <state_time_stamp> 1259936661 </state_time_stamp>
                                <job_specifics> <nprocs> 1 </nprocs> <disk> 2G </disk> <mem> 64G </mem> <repository>
LGI@lgislave.tc.lic.leidenuniv.nl:/var/www/html/LGI/repository/JOB_200f61825ff3f59c736d737612e25e3d </repository>
<repository_url> https://lgislave.tc.lic.leidenuniv.nl/LGI/repository/JOB_200f61825ff3f59c736d737612e25e3d
</repository_url> </job_specifics>
                        </job>
                </response>
</LGI>
```

The project server automatically sorts the jobs for the response according to the 'priority' field of the job and implements a basic FIFO scheduling algorithm as the field is initialized with the submit time stamp. A scheduler can change this to offer a different quality of service. See below.

The resource daemons should now inspect the individual jobs in the response and decide to accept the job or reject it. When the resource decides to reject it, it should unlock this job so other resources can have a look at it. If it does decide to run the job, it should change the state of the job to 'running' and request the details of the job as explained below.

4) The request '***resource_request_job_details***';

This request is made to the project server to retrieve the full information for a job identified through it's 'job_id'. The resource is again identified through the certificate and only if the resource is known and has a running lock on the job, this request is served. The 'job_id' is again POSTed together with the 'session_id' because running session with the resource is needed for this request to be able to serve. The response is very similar to the above described response, the extra information of the 'input', 'output', 'application', 'repository_content' and the 'state' are now also returned. The 'input' and 'output' blobs are encoded into a hexadecimal (BinHex) notation so they can be transferred in the XML format over https. An example response is:

```
<LGI>
        <CA_certificate> http://lgi.tc.lic.leidenuniv.nl/LGI/certificates/LGI-CA.crt </CA_certificate>
        <server_max_field_size> 65355 </server_max_field_size>
        <LGI_version> 1.34 </LGI_version>
        <API_version> 1.32 </API_version>
        <response>
                <resource> mark@testresource.somewere.nl </resource>
                <resource_url> </resource_url>
                <resource_capabilities> <hello_world> nothing </hello_world> </resource_capabilities>
                <project> testproject </project>
                <project_master_server> https://lgi.tc.lic.leidenuniv.nl/LGI </project_master_server>
                <this_project_server> https://lgislave.tc.lic.leidenuniv.nl/LGI </this_project_server>
                <session_id> 1 </session_id>
                <job>
                        <job_id> 147 </job_id>
                        <application> testapp </application>
                        <state> running </state>
                        <target_resources> any </target_resources>
                        <owners> sjoerd, cyttorn </owners>
                        <read_access> sjoerd, any </read_access>
                        <read_access> sjoerd </read_access>
                        <state_time_stamp> 1259936661 </state_time_stamp>
                        <job_specifics> <repository>
LGI@lgislave.tc.lic.leidenuniv.nl:/var/www/html/LGI/repository/JOB_000f61825ff3f59c736d737612e25e3d </repository>
<repository_url> https://lgislave.tc.lic.leidenuniv.nl/LGI/repository/JOB_000f61825ff3f59c736d737612e25e3d
</repository_url> </job_specifics>
                        <input> CDEF9021569C8787E </input>
                        <output> </output>
                        <repository_content> <file number="1"> <file_name> hi </file_name> <size> 1024 </size>
<date> 1662626 </date> <number_of_files> 1 </number_of_files> </file> </repository_content>
                </job>
        </response
</LGI>
```

5) The request '***resource_lock_job***';

This request is made to the project server to lock a job in the 'job_queue' table. The resource is identified through the certificate, the 'job_id' together with the 'session_id' is POSTed and the lock will only be set if the job has not been locked before by any other resource and if the job's 'target_resources' contains “any” or the requesting

resource name. If the resource is unknown, this service is again denied and an error response is returned. A successful respond will be in the “<lock> </lock>” tags wrapped XML, with details of the lock:

```
<LGI>
        <CA_certificate> http://lgi.tc.lic.leidenuniv.nl/LGI/certificates/LGI-CA.crt </CA_certificate>
        <server_max_field_size> 65355 </server_max_field_size>
        <LGI_version> 1.34 </LGI_version>
        <API_version> 1.32 </API_version>
        <response>
                <resource> mark@testresource.somewere.nl </resource>
                <resource_url> </resource_url>
                <resource_capabilities> <hello_world> nothing </hello_world> </resource_capabilities>
                <project> testproject </project>
                <project_master_server> https://lgi.tc.lic.leidenuniv.nl/LGI </project_master_server>
                <this_project_server> https://lgislave.tc.lic.leidenuniv.nl/LGI </this_project_server>
                <session_id> 103 </session_id>
                <job_id> 104 </job_id>
                <lock>
                        <lock_id> 10020 </lock_id>
                        <job_id> 104 </job_id>
                        <lock_time> 127657651276 </lock_time>
                        <session_id> 103 </session_id>
                        <resource_id> 3 </resource_id>
                </lock>
        </response>
</LGI>
```

If the lock cannot be set, an error response is sent back, as described previously. The resource should now wait and try again later.

6) The request '***resource_unlock_job***';

This request is made to the project server to unlock a previously set lock on a job. Again the resource is identified through the certificate and both the 'job_id' and the 'session_id' should be POSTed. The lock will only be removed if the resource is known and had originally locked the job, or a job was locked for the resource through the 'resource_request_work' request. The response of a successful call again is embedded into the “<lock> </lock>” tags:

```
<LGI>
        <CA_certificate> http://lgi.tc.lic.leidenuniv.nl/LGI/certificates/LGI-CA.crt  </CA_certificate>
        <server_max_field_size> 65355 </server_max_field_size>
        <LGI_version> 1.34 </LGI_version>
        <API_version> 1.32 </API_version>
        <response>
                <resource> mark@testresource.somewere.nl </resource>
                <resource_url> </resource_url>
                <resource_capabilities> <hello_world> nothing </hello_world> </resource_capabilities>
                <project> testproject </project>
                <project_master_server> https://lgi.tc.lic.leidenuniv.nl/LGI  </project_master_server>
                <this_project_server> https://lgislave.tc.lic.leidenuniv.nl/LGI </this_project_server>
                <session_id> 103 </session_id>
                <job_id> 104 </job_id>
                <lock>
                        <lock_id> 10020 </lock_id>
                        <job_id> 104 </job_id>
                        <unlock_time> 127657651276 </unlock_time>
                        <session_id> 103 </session_id>
                        <resource_id> 3 </resource_id>
                </lock>
        </response>
</LGI>
```

7) The request '***resource_update_job***';

This request is made to the server to alter some of the fields of a job. The job should be locked by the resource before this request is allowed. The resource should have a running session too. The 'job_id' and the 'session_id' should be POSTed together with fields that can be altered; 'state', 'target_resources', 'input', 'output' and 'job_specifics'. If any of these fields need changing, the new data should be POSTed. The server automatically adjusts the 'state_time_stamp' field if needed. The response of the server is the final data the job record will contain, as described by the 'resource_job_details' requests response. The 'input' and 'output' data POSTed should be first BinHexed and the server should un-BinHex them.

8) The request '***resource_submit_job***';

This request is made by a resource to insert a new job into the project server database. Again a running session is needed for this request. The fields needed for this request are 'session_id', 'state', 'application', 'owners' and 'target_resources'. The other fields, 'read_access', 'job_specifics', 'input' and 'output', can also be POSTed if needed.

The server will check if any of the POSTed 'owners' are allowed to submit a job for the specific 'application'. If not, an error response is returned. The 'owners' credentials are checked against the 'users_denied', 'groups_denied', 'users_allowed' and 'groups_allowed' tables. Only the subset of the 'owners' that is allowed to submit a job will become part of the jobs 'owners'.

If the credentials are in order, the job is inserted into the database and automatically will be locked by the project server for the resource, identified through the client certificate. The 'job_id' will be determined by and the 'state_time_stamp' will also be initialized by the server.

If the 'read_access' data was not POSTed, it will be set equal to the 'owners' by the server. The same applies to 'write_access'. During the submission of the job, this routine automatically creates a repository for the job. Files uploaded to the server are automatically moved into the repository directory and the correct job_specifics tags "<repository> </repository>" and "<repository_url> </repository_url>" are added. Read more about the repositories below. Files can be uploaded by POSTing the variable "number_of_uploaded_files" to specify the number of files being uploaded. Each file is given a POST name like "uploaded_file_XXX" with XXX being an integer.

The response is again equal to the response obtained from the server in the 'resource_job_details' request. The resource can check this data, if needed, change fields by issuing an 'resource_update_job' request and finally do the 'resource_unlock_job' request.

9) The request '***resource_job_state***';

This request is made by the resource to get the state of a job, identified through the 'job_id' being POSTed. This request is only served if the known resource, identified through the certificate, is present in the 'target_resources' list of the job, or if this field contains the "any" resource. The response by the server is very similar to the one obtained by issuing the 'resource_job_details' request. The difference is that the 'input' , 'output' and 'repository_content' fields are now not reported and that the job need not be locked prior to the request. Neither does the resource need to have a running session for this request. An example of a valid response is thus:

```
<LGI>
        <CA_certificate> http://lgi.tc.lic.leidenuniv.nl/LGI/certificates/LGI-CA.crt </CA_certificate>
        <server_max_field_size> 65355 </server_max_field_size>
        <LGI_version> 1.34 </LGI_version>
        <API_version> 1.32 </API_version>
        <response>
                <resource> mark@testresource.somewere.nl </resource>
                <resource_url> </resource_url>
                <resource_capabilities> <hello_world> nothing </hello_world> </resource_capabilities>
                <project> testproject </project>
                <project_master_server> https://lgi.tc.lic.leidenuniv.nl/LGI </project_master_server>
                <this_project_server> https://lgislave.tc.lic.leidenuniv.nl/LGI </this_project_server>
                <job>
                        <job_id> 147 </job_id>
                        <application> testapp </application>
                        <state> running </state>
                        <target_resources> any </target_resources>
                        <owners> sjoerd </owners>
                        <read_access> any </read_access>
                        <write_access> any </write_access>
                        <state_time_stamp> 1259936661 </state_time_stamp>
                        <job_specifics> <repository>
LGI@lgislave.tc.lic.leidenuniv.nl:/var/www/html/LGI/repository/JOB_000f61825ff3f59c736d737612e25e3d </repository>
```

```
<repository_url> https://lgislave.tc.lic.leidenuniv.nl/LGI/repository/e3d/JOB_000f61825ff3f59c736d737612e25e3d
</repository_url> </job_specifics>
                  </job>
         </response>
</LGI>
```

This request can be issued to monitor a change in state of a job by the resource. This can be used to detect an 'aborting' state issued by an application interface. This request also updates the 'daemon_pulse' field of a job.

10) The request '***resource_request_resource_data***';

This request is made by the resource to get the certificate details of another resource. The compulsory field 'resource_name' should be POSTed to be able to serve this request. The resource in again identified through it's certificate and is allowed to POST the 'project' field too. The certificate is reported in BinHexed format and no session is needed to serve this request. The response to this request looks like:

```
<LGI>
         <CA_certificate> http://lgi.tc.lic.leidenuniv.nl/LGI/certificates/LGI-CA.crt </CA_certificate>
         <server_max_field_size> 65355 </server_max_field_size>
         <LGI_version> 1.34 </LGI_version>
         <API_version> 1.32 </API_version>
         <response>
                  <resource> mark@testresource.somewere.nl </resource>
                  <resource_url> mark@testresource.somewere.nl </resource_url>
                  <resource_capabilities> <hello_world> nothing </hello_world> </resource_capabilities>
                  <project> testproject </project>
                  <project_master_server> https://lgi.tc.lic.leidenuniv.nl/LGI </project_master_server>
                  <this_project_server> https://lgislave.tc.lic.leidenuniv.nl/LGI </this_project_server>
                  <requested_resource_data>
                           <resource_name>  mark@othertestresource.somewere.nl </resource_name>
                           <client_certificate> 0F1206AC5D ... 1B4D6E </client_certificate>
                           <resource_url> mark@othertestresource.somewere.nl <resource_url>
                           <last_call_time> 1287876287 </last_call_time>
                  </requested_resource_data>
         </response>
</LGI>
```

Job locking mechanism:

Because in the LGI several resources can request work form the project server, and decide for them selves, based on the 'job_specifics' field to perhaps accept the work for the application, a locking mechanism is needed. If a resource decides to inspect the job, another resource should not in the mean-time decide to run the job or else a race-condition is met.

The locking mechanisms in the LGI will be implemented per job and a resource is only allowed to work on a job entry of a project database if the resource has successfully locked that specific job prior to use. Moreover, the project's web server will refuse to serve 'resource_request_work' requests to the resource if a lock exists in the 'running_locks' table for that resource session. If somehow a resource fails to unlock the job after a while, the project servers will unlock the jobs that where locked and kill the session it had and the resource should setup a new session with the project server by signing up again.

When the 'resource_request_work' request is made, the project's server follows the following steps to ensure a correct and safe locking mechanism; The requesting resource name is taken from the client-certificate it presents. It will check if the resource is registered and had a valid session running. If the resource was not registered or has no valid session, an error response is returned. Otherwise, a query is run on the 'running_locks' table for the project to see if there are any lock currently running for that resource in the current session. If so, again an error is returned.

If no locks were present on any jobs for this session and resource, the 'job_queue' table is queried for the specific resource for unlocked jobs (lock_state<=0), for the requested

application, being in the 'queued' state. A limit is set to this request and results are sorted according to the 'priority' field. For each job a lock is now requested for this resource by increasing the 'lock_state' field, by trying to insert a 'running_lock' entry for this resource, session and job (the 'job_id' field is UNIQUE in the 'running_locks' table). It will also set the current time as the 'lock_time' in the 'running_locks' table. If the insert fails, the 'lock_state' is decreased again and the job is then ignored and not put into the response. If the insert succeeded, a lock was acquired. The job is then inspected more thoroughly by the 'resource_request_work' code to see if no other resource has already claimed this job (state is still 'queued', this resource is still allowed and listed in the 'target_resources' field and the 'lock_state' field is equal to 1. If it has not been claimed yet, it will be put into the response. If it was claimed already, it will be ignored and unlocked again. This locking mechanism avoids table locks that thwart scaling and concurrency.

When a request is made by a resource to unlock a job, the resource is first identified and checked if it had a lock on the job in this session. If it did, the corresponding entry in the 'running_locks' table is removed before the 'lock_state' of the job in the 'job_queue' is decreased.

Because of the extensive locking mechanisms implemented in LGI, no replication and MariaDB/MySQL proxy load-balancing techniques should be used. See below for more details when a project needs to scale up. Also special effort has been taken to make the locking algorithms scale for concurrency by avoiding full table locks and optimizing the number of queries needed.

Application interface interfacing with the project server:

In this section the communication between an application interface and the project servers is detailed upon. In general the URL used to communicate between a resource and the project server can be different than the URL used by the application interfaces that also communicate with the project server. For resources there is no need to check the user and group access control lists of the database (the 'user_allowed', 'user_denied', 'group_allowed', 'group_denied' tables), a resource is identified through it's certificate, usually resources interact differently and perhaps more frequent with the project server than users do. Moreover, by logically separating the web-servers for resources and application interfaces, extra scaling possibilities are introduced.

In general, all communications between the application interface and the project server need user credentials in the request. Also all communication is done in XML over https, using client certificates, signed by the LGI-CA, but now with common-names fields within the certificates being equal to the user credentials. See the later section on certificates for details on this.
The user credentials need to be checked for each request and only jobs, for which the user has access to, through the 'owners', 'write_access' and 'read_access' fields, can be changed and inspected respectively.

Again all responses by the server are to be wrapped into the “<LGI> <response> </response> </LGI>” tags. If an error condition is met, the response will contain the usual “<error> </error>” tags embedded XML messages.

The following requests can be made by the application interface to the project server. They are listed under the 'interfaces' directory on the project server's main URL. For example “https://lgi.tc.lic.leidenuniv.nl/LGI/interfaces/interface_submit_job.php”.

1) The '***interface_submit_job***' request;

This request is done by an application interface to submit a job. The user is identified through the certificate's common-name field and by the POSTed 'user' and 'groups' fields. The POSTed 'user' field should equal the certificates common-name or otherwise service is denied. Further certificate checks that might be done are described below. The other compulsory fields that also need to be POSTed in this request are the 'application' and the 'target_resources' fields. The 'input', the 'job_specifics', the 'write_access' and 'read_access' fields are not compulsory but can be POSTed too.

If both the 'write_access' and 'read_access' fields were not POSTed, they will be set to the 'users' field. The 'target_resources' is checked to contain valid resources, known to the project and with the 'application' configured, or the “any” resource. The 'application' field is also checked for in the database. The 'input' field, when needed, is again POSTed in BinHex format, which is again un-BinHexed by the server.

A new job will created by the server, based on the above described values of POSTed fields. The default state of the job will be the 'queued' state. During the submission of the job, this routine automatically creates a repository for the job. Files uploaded to the server are automatically moved into the repository directory and the correct job_specifics tags “<repository> </repository>” and “<repository_url> </repository_url>” are added. Read more about the repositories below. Files can be uploaded by POSTing the variable “number_of_uploaded_files” to specify the number of files being uploaded. Each file is given a POST name like “uploaded_file_XXX” with XXX being an integer. Clearly the other fields will be set by the server and a response to the application interface will contain the POSTed details:

```
<LGI>
        <CA_certificate> http://lgi.tc.lic.leidenuniv.nl/LGI/certificates/LGI-CA.crt </CA_certificate>
        <server_max_field_size> 65355 </server_max_field_size>
        <LGI_version> 1.34 </LGI_version>
        <API_version> 1.32 </API_version>
        <response>
                <project> testproject </project>
                <project_master_server> https://lgi.tc.lic.leidenuniv.nl/LGI </project_master_server>
                <this_project_server> https://lgislave.tc.lic.leidenuniv.nl/LGI </this_project_server>
                <user> mark </user>
                <groups> teras, cyttron </groups>
                <job>
                        <job_id> 147 </job_id>
                        <application> testapp </application>
                        <state> queued </state>
                        <target_resources> any </target_resources>
                        <owners> sjoerd, cyttron </owners>
                        <read_access> any, sjoerd, cyttron </read_access>
                        <write_access> sjoerd, cyttron </write_access>
                        <state_time_stamp> 1259936661 </state_time_stamp>
                        <job_specifics> <nprocs> 2 </nprocs> <disk> 5G </disk> <mem> 64G </mem> <repository>
LGI@lgislave.tc.lic.leidenuniv.nl:/var/www/html/LGI/repository/JOB_000f61825ff3f59c736d737612e25e3d </repository>
<repository_url> https://lgislave.tc.lic.leidenuniv.nl/LGI/repository/e3d/JOB_000f61825ff3f59c736d737612e25e3d
</repository_url> </job_specifics>
                        <repository_content> <file number=”1”> <file_name> hi </file_name> <size> 1024 </size>
<date> 1662626 </date> </file> <number_of_files> 1 </number_of_files> </repository_content>
                        <input> CDEF9021569C8787E </input>
                </job>
        </response>
</LGI>
```

2) The '***interface_job_state***' request;

This request is made by the application interface to obtain information on the queue contained in the project server database. Again the 'user' and 'groups' field should be POSTed and the common-name of the client certificate should match the POSTed 'user' field. This request has two ways of generating an response, depending on whether the 'job_id' field is POSTed or not. In case the 'job_id' is not POSTed, the response will be a list of current jobs in the database, based on a selection by the possible POSTed 'application' and 'state' fields. Only jobs in which the POSTed 'user' or any of the 'groups' is part of the 'read_access' will be returned. In the response, the number of listed jobs will be wrapped into the “<number_of_jobs>

</number_of_jobs>” tags. A valid response looks like:

```
<LGI>
        <CA_certificate> http://lgi.tc.lic.leidenuniv.nl/LGI/certificates/LGI-CA.crt </CA_certificate>
        <server_max_field_size> 65355 </server_max_field_size>
        <LGI_version> 1.34 </LGI_version>
        <API_version> 1.32 </API_version>
        <response>
                <project> testproject </project>
                <user> mark </user>
                <groups> teras, cyttron </groups>
                <project_master_server> https://lgi.tc.lic.leidenuniv.nl/LGI </project_master_server>
                <this_project_server> https://lgislave.tc.lic.leidenuniv.nl/LGI </this_project_server>
                <number_of_jobs> 3 </number_of_jobs>

            <job number='1'>
                        <job_id> 140 </job_id>
                        <state> queued </state>
                        <application> testapp </application>
                        <state_time_stamp> 1259926661 </state_time_stamp>
                        <target_resources> testresource </target_resources>
                        <owners> mark </owners>
                        <read_access> theor, sara, cyttron </read_access>
                        <write_access> mark, theor </write_access>
                        <state_time_stamp> 1259926661 </state_time_stamp>
                        <job_specifics> <nprocs> 2 </nprocs> <disk> 5G </disk> <mem> 64G </mem> <repository>
LGI@lgislave.tc.lic.leidenuniv.nl:/var/www/html/LGI/repository/JOB_000f61825ff3f59c736d737612e25e3d </repository>
<repository_url> https://lgislave.tc.lic.leidenuniv.nl/LGI/ repository/JOB_000f61825ff3f59c736d737612e25e3d
</repository_url> </job_specifics>
                    </job>

            <job number='2'>
                        <job_id> 142 </job_id>
                        <state> running </state>
                        <application> testapp </application>
                        <state_time_stamp> 1259927661 </state_time_stamp>
                        <target_resources> testresource </target_resources>
                        <owners> mark </owners>
                        <read_access> theor, sara, cyttron </read_access>
                        <write_access> mark, theor </write_access>
                        <state_time_stamp> 1259926661 </state_time_stamp>
                        <job_specifics> <nprocs> 3 </nprocs> <disk> 4G </disk> <mem> 64G </mem> <repository>
LGI@lgislave.tc.lic.leidenuniv.nl:/var/www/html/LGIrepository/JOB_010f61825ff3f59c736d737612e25e3d </repository>
<repository_url> https://lgislave.tc.lic.leidenuniv.nl/LGI/ repository/JOB_010f61825ff3f59c736d737612e25e3d
</repository_url> </job_specifics>
            </job>

            <job number='3'>
                        <job_id> 147 </job_id>
                        <state> finished </state>
                        <application> testapp </application>
                        <state_time_stamp> 1259936661 </state_time_stamp>
                        <target_resources> testresource </target_resources>
                        <owners> mark </owners>
                        <write_access> mark, theor </write_access>
                        <read_access> theor, sara, cyttron </read_access>
                        <state_time_stamp> 1259926661 </state_time_stamp>
                        <job_specifics> <nprocs> 1 </nprocs> <disk> 1G </disk> <mem> 64G </mem> <repository>
LGI@lgislave.tc.lic.leidenuniv.nl:/var/www/html/LGI/repository/JOB_100f61825ff3f59c736d737612e25e3d </repository>
<repository_url> https://lgislave.tc.lic.leidenuniv.nl/LGI/repository/JOB_100f61825ff3f59c736d737612e25e3d
</repository_url> </job_specifics>
                </job>
        </response>
</LGI>
```

In this mode, it does not matter if the jobs reported have running locks or not. If the POSTed 'limit' field has a value of 0, a count of the number of jobs found is returned only. Also, in this mode, the “<repository_content> </repository_content>”, “<input> </input>” and “<output> </output>” information is not present.

If the request was made with a specific 'job_id' field POSTed, the response will be more elaborate for the specific job only:

```
<LGI>
        <CA_certificate> http://lgi.tc.lic.leidenuniv.nl/LGI/certificates/LGI-CA.crt </CA_certificate>
        <server_max_field_size> 65355 </server_max_field_size>
        <LGI_version> 1.34 </LGI_version>
        <API_version> 1.32 </API_version>
        <response>
                <project> testproject </project>
                <user> mark </user>
                <groups> teras, cyttron </groups>
                <project_master_server> https://lgi.tc.lic.leidenuniv.nl/LGI </project_master_server>
                <this_project_server> https://lgislave.tc.lic.leidenuniv.nl/LGI </this_project_server>
                <number_of_jobs> 1 </number_of_jobs>
                <job_id> 140 <job_id>

            <job number=1>
                        <job_id> 140 </job_id>
                        <state> queued </state>
                        <application> testapp </application>
                        <target_resources> testresource </target_resources>
                        <owners> mark </owners>
                        <read_access> theor, sara, cyttron </read_access>
                        <state_time_stamp> 1259926661 </state_time_stamp>
                        <job_specifics> <nprocs> 2 </nprocs> <disk> 5G </disk> <mem> 64G </mem> <repository>
LGI@lgislave.tc.lic.leidenuniv.nl:/var/www/html/LGI/repository/JOB_000f61825ff3f59c736d737612e25e3d </repository>
<repository_url> https://lgislave.tc.lic.leidenuniv.nl/LGI/repository/e3d/JOB_000f61825ff3f59c736d737612e25e3d
</repository_url> </job_specifics>
                        <input> DCEF18982 </input>
                        <output> </output>
                        <repository_content> <file number=”1”> <file_name> hi </file_name> <size> 1024 </size>
```

```
<date> 1662626 </date> </file> <number_of_files> 1 </number_of_files> </repository_content>
                </job>
        </response>
</LGI>
```

Now the “<input> </input>”, “<output> </output>” and “<repository_content> </repository_content>” information is included in the response.

If the possible POSTed 'job_id' is not present in the database, or the 'user', or any of his 'groups' is not part of the 'read_access' fields of the job, service is denied and an error response is returned. Also if the specific job has a lock running, an error response is sent back after the time-out period of 30 seconds has passed. During this time-out period the server waits for the lock to be cleared before a response is generated and sent back.

3) The ***'interface_delete_job***' request;

This request is made by an application interface to remove a job from the project server database. Again the 'user' and 'groups' fields should be POSTed and the 'user' field should match the certificate. Now also POSTed is the 'job_id' of the job that is to be killed or deleted. If the 'user', or any of the 'groups' is not part of the 'write_access' field of the job with the POSTed 'job_id', service is denied and an error response is sent back.

If the user credentials match and the job with the POSTed 'job_id' is in the 'finished', 'aborted' or 'queued' state, the job can be safely removed from the database if no locks are present on the job. If the job was locked an error response is sent back after a certain time-out. During the time-out of 30 seconds, the server waits for the lock to be freed first.

If the job was in any other state than 'queued' or 'finished', the job is set into the 'aborting' state and not removed from the database, only if no lock is active on the job. The 'aborting' state allows the resource that is currently dealing with the job, to detect the abort signal. The resource will then set the job into the 'finished' or 'aborted' state when possible. Whether the resource can abort a job immediately is highly dependent on the application and the resource it is running on. Again, if the job is still locked after the preset time-out period of 30 seconds, an error response is sent back.

If the job was successfully removed from the database, or has been set into the 'aborting' state, the response of the servers is similar to the response obtained by the 'user_job_state' request described above:

```
<LGI>
        <CA_certificate> http://lgi.tc.lic.leidenuniv.nl/LGI/certificates/LGI-CA.crt  </CA_certificate>
        <server_max_field_size> 65355 </server_max_field_size>
        <LGI_version> 1.34 </LGI_version>
        <API_version> 1.32 </API_version>
        <response>
                <user> mark </user>
                <groups> teras cyttron </groups>
                <project> testproject </project>
                <project_master_server> https://lgi.tc.lic.leidenuniv.nl/LGI </project_master_server>
                <this_project_server> https://lgislave.tc.lic.leidenuniv.nl/LGI </this_project_server>
            <job>
                        <job_id> 140 </job_id>
                        <state> aborting </state>
                        <application> testapp </application>
                        <target_resources> testresource </target_resources>
                        <owners> mark </owners>
                        <read_access> theor, sara, cyttron </read_access>
                        <write_access> theor </write_access>
                        <state_time_stamp> 1259926661 </state_time_stamp>
                        <job_specifics> <repository>
LGI@lgislave.tc.lic.leidenuniv.nl:/var/www/html/LGI/repository/JOB_000f61825ff3f59c736d737612e25e3d </repository>
<repository_url> https://lgislave.tc.lic.leidenuniv.nl/LGI/repository/e3d/JOB_000f61825ff3f59c736d737612e25e3d
</repository_url> </job_specifics>
                </job>
        </response>

</LGI>
```

This response allows the application interface to record or log jobs if desired.

4) The '***interface_project_server_list***' request;

This request is made to get a list of project servers for a specific project. The 'user' and 'groups' field are POSTed and compared to the clients-certificates common-name. If they do not match an error response is returned and service is denied. If the credentials are met, a valid response will contain a list of project servers:

```
<LGI>
        <CA_certificate> http://lgi.tc.lic.leidenuniv.nl/LGI/certificates/LGI-CA.crt </CA_certificate>
        <server_max_field_size> 65355 </server_max_field_size>
        <LGI_version> 1.34 </LGI_version>
        <API_version> 1.32 </API_version>
        <response>
                <user> mark </user>
                <groups> teras cyttron </groups>
                <project> testproject </project>
                <project_master_server> https://lgi.tc.lic.leidenuniv.nl/LGI </project_master_server>
                <this_project_server> https://lgislave.tc.lic.leidenuniv.nl/LGI </this_project_server>
                <number_of_slave_servers> 2 </number_of_slave_servers>
                <project_server number='1'> https://lgislave.tc.lic.leidenuniv.nl/LGI </project_server>
                <project_server number='2'> https://lgislave2.tc.lic.leidenuniv.nl/LGI </project_server>
        </response>
</LGI>
```

The application interface can now choose what project server to communicate to to submit or query about jobs.

5) The '***interface_project_resource_list***' request;

This request is made to get a list of resources available for a specific project. The 'user' and 'groups' field are POSTed and compared to the clients-certificates common-name. If they do not match an error response is returned and service is denied. If the credentials are met, a valid response will contain a list of resources with some of the details on each resource:

```
<LGI>
        <CA_certificate> http://lgi.tc.lic.leidenuniv.nl/LGI/certificates/LGI-CA.crt  </CA_certificate>
        <server_max_field_size> 65355 </server_max_field_size>
        <LGI_version> 1.34 </LGI_version>
        <API_version> 1.32 </API_version>
        <response>
                <user> mark </user>
                <groups> teras cyttron </groups>
                <project> testproject </project>
                <project_master_server> https://lgi.tc.lic.leidenuniv.nl/LGI </project_master_server>
                <this_project_server> https://lgislave.tc.lic.leidenuniv.nl/LGI </this_project_server>
                <number_of_resources> 2 </number_of_resources>
                <resource number='1'>
                        <resource_name> mark@lgi.tc.lic.leidenuniv.nl </resource_name>
                        <resource_capabilities> <hello_world> </hello_world> </resource_capabilities>
                        <last_call_time> 1990282828 </last_call_time>
                </resource>
                <resource number='2'>
                        <resource_name> somers@cartesius.surfsara.nl </resource_name>
                        <resource_capabilities> <hello_world> </hello_world> </resource_capabilities>
                        <last_call_time> 1990272828 </last_call_time>
                </resource>
        </response>
</LGI>
```

The application interface can now choose what resource to target.

Project server synchronization API:

In this part of the document the API that deals with the communication between two or more project servers within a single project is detailed upon. This API has been developed to allow for project servers to exchange updates to keep all servers within a project synchronized.

Each project server within a project is identified as being a resource that has the

'project_server' field set to non-zero in the 'active_resources' table of all the project servers. A project server is therefore also identified through a resource client-certificate signed by the LGI-CA.

Updates are queries that can be executed on the project server's database. Each update is identified by an integer 'version' number and only sequential updates are allowed. Each update will be logged into the 'updates' table.

Again all responses by the server are to be wrapped into the "<LGI> <response> </response> </LGI>" tags. If an error condition is met, the response will contain the usual "<error> </error>" tags embedded XML messages.

The following requests can be made by any other project server within the project to the project server. They are listed under the 'servers' directory on the project server's main URL. For example "https://lgi.tc.lic.leidenuniv.nl/LGI/servers/server_get_update.php".

1) The '***server_get_update***' request;

This request allows for any project server to inquire this server for the presence of possible updates. The requesting server is identified through it's certificate common name. Only if the requesting server is listed as a project server, this request can be met. An error response is returned otherwise.
The compulsory 'version' and 'project' fields are POSTed and identify the requesting server's latest update. If the server has any updates with a higher version number in the 'updates' table, it will report the first sequentially applicable update the requesting server should apply or log into it's own 'updates' table. A valid response looks like:

```
<LGI>
        <CA_certificate> http://lgi.tc.lic.leidenuniv.nl/LGI/certificates/LGI-CA.crt  </CA_certificate>
        <server_max_field_size> 65355 </server_max_field_size>
        <LGI_version> 1.34 </LGI_version>
        <API_version> 1.32 </API_version>
        <response>
                <project> testproject </project>
                <project_master_server> https://lgi.tc.lic.leidenuniv.nl/LGI </project_master_server>
                <this_project_server> https://lgislave.tc.lic.leidenuniv.nl/LGI </this_project_server>
                <requesting_project_server> https://lgi.tc.lic.leidenuniv.nl/LGI </ requesting_project_server>
                <requesting_project_server_version> 0 </requesting_project_server_version>
                <update>
                        <updates> 2 </updates>
                        <update_version> 1 </update_version>
                        <target_servers> any </target_servers>
                        <update_query> 0002020010002AB3 </update_query>
                </update>
        </response>
</LGI>
```

If there are no relevant updates, the above response will contain "<updates> 0 </updates>" and the latest version number encountered in the database is reported within the "<update_version> </update_version>" tags.

2) The '***server_run_update***' request;

This request allows for any other project server within the project to push an update to this server. For this to work the 'version', 'servers' and 'update' fields together with the 'project' fields should be POSTed. If the server has any updates higher than the POSTed version, it will ignore this update being pushed and return it's highest version number within the "<update_version> </update_version>" tags. If this server should sequentially apply or log this update, it will do so. If this server has to apply other updates first, an update cycle is started. If the 'version', 'servers' and 'update' fields were not POSTed, also an update cycle is started. A valid response, when an update is being pushed, looks like:

```
<LGI>
        <CA_certificate> http://lgi.tc.lic.leidenuniv.nl/LGI/certificates/LGI-CA.crt  </CA_certificate>
        <server_max_field_size> 65355 </server_max_field_size>
        <LGI_version> 1.34 </LGI_version>
        <API_version> 1.32 </API_version>
        <response>
                <project> testproject </project>
                <project_master_server> https://lgi.tc.lic.leidenuniv.nl/LGI </project_master_server>
                <this_project_server> https://lgislave.tc.lic.leidenuniv.nl/LGI </this_project_server>
                <requesting_project_server> https://lgi.tc.lic.leidenuniv.nl/LGI </ requesting_project_server>
                <update_version> 1 </update_version>
                <target_servers> any </target_servers>
                <update_query> 0102030405011 </update_query>
                <update>
                        <update_version> 1 </update_version>
                </update>
        </response>
</LGI>
```

During an update cycle, each of the marked servers is requested to report back any possible update. For this a POST to the 'server_get_update' API routine is performed on each server. If all updates have been applied and all project servers have been polled, the update cycle is finished. A valid response after such an update cycle looks like:

```
<LGI>
        <CA_certificate> http://lgi.tc.lic.leidenuniv.nl/LGI/certificates/LGI-CA.crt </CA_certificate>
        <server_max_field_size> 65355 </server_max_field_size>
        <LGI_version> 1.34 </LGI_version>
        <API_version> 1.32 </API_version>
        <response>
                <project> testproject </project>
                <project_master_server> https://lgi.tc.lic.leidenuniv.nl/LGI </project_master_server>
                <this_project_server> https://lgislave.tc.lic.leidenuniv.nl/LGI </this_project_server>
                <requesting_project_server> https://lgi.tc.lic.leidenuniv.nl/LGI </ requesting_project_server>
                <update>
                        <update_version> 1 </update_version>
                </update>
        </response>
</LGI>
```

Certificates:

Within the LGI heavy use is made of x509 server- and client-certificates. This part of the document details on the 'CommonName' field of these certificates and how they are used within the LGI. For actually managing your project certificates one could use easy-rsa from the EPEL repository.

For certificates belonging to resources the 'CommonName' contains possibly two fields separated by a ';'. The first field specified the resource unique name which is also present in the 'active_resources' table. The naming convention for resources is “user@machine.org”. The second field is a comma separated field of all the projects this resource has access to. These fields within the certificate always take precedence to any other access rules implemented in the project server or fields posted to the project server. Resources always supply a valid client-certificate to the project servers.

For certificates belonging to users or application interfaces, the 'CommonName' possibly contains three fields separated by a ';'. The first field is the name of the user that accesses the project server through the application interfaces API. Also for interfaces and users the naming convention is “username@location.org”. The possible present second field is either a comma separated field with all the groups the user is allowed to belong to or again a comma separated field with all the projects the user is allowed to interface to. If only two fields separated by a ';', the second field is the list of projects the is allowed to interface to. Application interface's always supply a valid user's client-certificate to the project servers.

Certificates used when configuring Apache for project servers are just normal x509 certificates in which the 'CommonName' has the correct fully qualified hostname of the server. The scheduler however uses another x509 certificate with the same convention for the 'CommonName' as a resource certificate. Read 'SETUP.txt' for more information on this.

It is however possible for resources and project server-certificates to only use the fully qualified hostname as the commonname. Chosing to do so, resource management through the x509 certificates is not possible anymore and it is also not possible anymore for a single host to run several independent resource deamons with unique identities. Multiple resource daemon instances on a single host can however run independently using the same simple hostname based identity if and only if they all have the same 'capabilities' and each monitor a separate 'run_directory' directory. See below.

Resource daemon:

In this section a more detailed description of the resource daemon is given, which by default, runs as a non privileged user on the resource. The resource daemon is configured by a configuration file. This file has also uses the XML format and the workings of the resource daemon is best described using the following example configuration:

```
<LGI>
        <ca_certificate_file> ../certificates/LGI+CA.crt </ca_certificate_file>
        <resource>
                <resource_certificate_file> ../certificates/laptop.crt </resource_certificate_file>
                <resource_key_file> ../certificates/laptop.key </resource_key_file>
             <run_directory> ./runhere </run_directory>

            <owner_allow> <any> 10 </any> </owner_allow>
                <owner_deny> </owner_deny>
                <job_limit> 20 </job_limit>

                <number_of_projects> 1 </number_of_projects>

                <project number='1'>
                        <project_name> LGI </project_name>
                        <project_master_server> https://lgi.tc.lic.leidenuniv.nl/LGI </project_master_server>

                  <owner_allow> <any> 5 </any> </owner_allow>
                        <owner_deny> </owner_deny>
                        <job_limit> 10 </job_limit>

                        <number_of_applications> 1 </number_of_applications>

                        <application number='1'>
                                <application_name> hello_world </application_name>

                                <owner_allow> <any> 1 </any> </owner_allow>
                                <owner_deny> </owner_deny>
                                <job_limit> 4 </job_limit>
                                <max_output_size> 4096 </max_output_size>

                                <job_sandbox_uid> 50001 </job_sandbox_uid>
                                <capabilities> nothing special </capabilities>

                                <check_system_limits_script>
./hello_world_scripts/check_system_limits_script </check_system_limits_script>
                                <job_check_limits_script> ./hello_world_scripts/job_check_limits_script
</job_check_limits_script>
                           <job_check_running_script> ./hello_world_scripts/job_check_running_script
</job_check_running_script>
                                <job_check_finished_script> ./hello_world_scripts/job_check_finished_script
</job_check_finished_script>
                                <job_prologue_script> ./hello_world_scripts/job_prologue_script
</job_prologue_script>
                                <job_run_script>./hello_world_scripts/job_run_script </job_run_script>
                                <job_epilogue_script> ./hello_world_scripts/job_epilogue_script
</job_epilogue_script>
                                <job_abort_script> ./hello_world_scripts/job_abort_script
</job_abort_script>
                        </application>
                </project>

        </resource>
</LGI>
```

The resource daemon reads the the above illustrated configuration from the supplied configuration file. Before running any further, the daemon verifies all the tags of the configuration. Files referred to should be readable and exist, directories specified should be readable and also exist, all the above illustrated tags should be present and so on.

After the daemon has verified the configuration, the configuration is stored in memory and used as long as the daemon is active. Changes to the configuration file can be made active by restarting the daemon. To stop the daemon, when it is running in the

background, a simple 'kill' command can be used to let the signal handler of the daemon to gracefully shut down the daemon. Any open sessions to any server are then guaranteed to be finished before the daemon is stopped.

The daemon will also generate a log. The logging level of the daemon can be supplied at the command line when the daemon is started. At the same time, the configuration file is supplied together with the option whether or not to daemonize and run in the background. If the daemon receives a SIGHUP signal, it will reload the configuration file and restart the logging.

Below are listed the options that the daemon supports:

```
./LGI_daemon.x [options] configfile

options:

-d           daemonize and run in background.
-q           log only critical messages.
-ft time     specify fast schedule cycle time in seconds. default is 120.
-st time     specify slow schedule cycle time in seconds. default is 600.
-n           log normal messages. this is the default.
-v           also log debug messages.
-vv          also log verbose debug messages.
-W           be less strict in hostname checks of project server certificates.
-l file      use specified logfile. default is to log to standard output.
```

The daemon can run as any user on the system and it is preferred to run the daemon as an unprivileged user and not as root. If, however, the daemon is run as root, the sandboxing of jobs is automatically enabled. In general a random special uid will be used per job. Any script run for a job is correctly setuid to the random uid. One can also override the randomization of uids per application by using the "<job_sandbox_uid> </job_sandbox_uid>" tags in the configuration file.

In general, when the daemon has verified the configuration and is ready to run, the 'run_directory' directory is scanned through for cached jobs. This run directory contains a tree of projects and applications and within each branch, the jobs for that project and that application are cached on disk. Each job will have a unique job directory in which the complete state of the job is stored. If the daemon runs as root and the random uid sandboxing is used, care should be taken that the 'run_directory' is world readable for any user. The daemon will make sure that the unique job sub-directories are only accessible by the randomly chosen uid user.

The state of a job in its corresponding job directory is described by several files, all beginning with an 'LGI_' prefix. Each such file contains exactly the same information that is also stored in the projects web-server's database for that job or contains an exact copy of the scripts that should be used for that application, as was specified in the configuration file. Moreover, all the 'LGI_xxxxxx' files, except for the LGI_output file, each has a corresponding '.hash' file. In that file the hash fingerprint in hexadecimal format is stored. If any of the files or scripts are corrupted, edited or changed, the hash will make sure that this is detected. Scripts, for which the hash does not match, will not be run. Job directories that contain a file for which the hash does not match, will not be accepted and monitored. The following LGI_xxx files can be used by the job scripts: 'LGI_project', 'LGI_this_project_server', 'LGI_project_master_server', 'LGI_application', 'LGI_job_id', 'LGI_owners', 'LGI_read_access', 'LGI_write_access', 'LGI_job_specifics', 'LGI_target_resources', 'LGI_input', 'LGI_output', 'LGI_state', 'LGI_state_time_stamp', 'LGI_job_check_limits_script', 'LGI_job_check_running_script', 'LGI_job_check_finished_script', 'LGI_job_prologue_script', 'LGI_job_run_script', 'LGI_job_epilogue_script', 'LGI_job_abort_script', 'LGI_key_file', 'LGI_certificate_file', 'LGI_ca_certificate_file', 'LGI_max_output_size', 'LGI_job_sandbox_uid', 'LGI_job_run_script_pid' and 'LGI_daemon_reference'. Of these only 'LGI_output' is allowed to be written to.

At the start of the daemon the run directory is scanned through for cached jobs. Any

job directory encountered and verified to be valid through the hashes, will be included into the set of jobs that are to be monitored by the daemon. Only one instance of the daemon can monitor a single run directory. When the daemon runs, every two minutes the state of the jobs that are to be monitored, are checked by contacting the correct job project server. If the job was issued on a slave server, only that slave server is contacted by using the 'resource_job_state' function. If, during this update cycle, the state in the database was changed, the job details are requested and the job directory on disk is synchronized to the result of the post.

If, after an update cycle, the daemon reference stored in 'LGI_daemon_reference' does not match the '<daemon_reference> </daemon_reference>' tags in the 'job_specifics', this job is considered to be taken over by another daemon. The jobs scripts will be used to gracefully kill the job. In this case the output and result of the job is ignored and the job directory is cleared.

After having updated all jobs on disk, each job will be inspected through the 'job_check_XXXX' scripts. These scripts were provided in the configuration file, but are now also stored, with a hash, in the job directory too. This allows for a successful completion of jobs even after the configuration file has changed the scripts for any of these applications.

In general the '***job_check_running_script***' script is started to check if the job is still running. This script should return an exit code of zero to signal that the job is still running. If the job was not found to be running, the '***job_check_finished_script***' is started to check if the job was finished. It also should return an exit code of zero if the job was finished. If the job was found to be running, the 'state' of the job is checked. If the 'state' of the job was changed to 'aborting', the '***job_abort_script***' script is run. If then the job_abort_script returns an exit code of zero, the job is considered to be aborted and the job details are again posted to the project server through the 'resource_update_job' function. After this successful post, the job is removed from the list of jobs to be monitored and the corresponding job directory is deleted. The job will end up in the 'aborted' state in the database on the project server. If the job was not found to be finished or running, the run sequence of the job will be started; first the '***job_prologue_script***' is started and if that script returns an exit code of zero, the '***job_run_script***' is started on the background.

If the job was found to be finished, because the 'job_check_finished_script' returned an exit code of zero, the '***job_epilogue_script***' script is started. If the 'job_epilogue_script' script returns an exit code of zero, the job details will be posted to the server. Again after a successful post, the job directory is removed, the job will not be monitored again and the job will end up in the 'finished' state on the project server's database.

The use of these 'job_XXXXX_scripts' allows the LGI resource user to fully configure the way the application should be started and handled. Moreover, these scripts can access any of the job details through the 'LGI_xxxxx' files also stored in the job directory. Because these scripts are copied and hashed into the job directory, a change in the configuration, or in the job scripts, will not affect any jobs prior to these changes.

Apart from doing the above described job update cycle every two minutes, the daemon also performs work request cycles every ten minutes or so. During this cycle all the projects and applications that are specified in the configuration file are inspected. The '***check_system_limits_script***' script is used to see if there is a system wide limit reached to running a job of this application. The script returns zero if no limit is present and work can be requested by posting to the 'resource_request_work' function

on the project server.

All jobs offered by any of the project servers are inspected and checked against the limits imposed by the 'owner_allow' and 'owner_deny' tags that were specified in the configuration file. If any of the jobs 'owners' was specified in any of the 'owners_deny' tags, the job is not accepted. If a limit was specified for any of the 'owners' in the 'owner_allow', by wrapping the maximum number of jobs allowed for that owner in tags of the owner: “<owner_allow> <mark> 2 </mark> <any> 1 </any> </owner_allow>”, the number of jobs currently being monitored is checked. If the set limit would be exceeded, the job is ignored again. Only if none of the owners would exceed any limits, the job will be checked further by running the '***job_check_limits_script***' script. The script will be run in a temporary job directory. If this script returns an exit code of zero, no limit is present for the job and the job will be accepted. The job will now be monitored and updated through the job update cycle of the daemon. If the job didn't get accepted, the temporary job directory is removed.

For projects with multiple project (slave) servers, a list of servers is compiled first during a work request cycle. Each of these servers is then requested to give work. The list of project (slave) servers is obtained by first signing up at the server specified by the 'project_master_server' tags in the configuration file.

Example scripts are included for the '**hello_world**' application configured to be run through several different resource managers on a cluster (SLURM, Torque / PBS, Load Leveler, SGE and gLite). The example configuration has been setup with the above scripts using no resource manager backend. Check the 'LGI.cfg' configuration file and the other 'hello_world' scripts for even more examples on how to configure the daemon or use different backends.

The actual daemon code is implemented in C++ and makes use of the Standard Template Library (STL) and the cURL library. The source code can be found in the 'src' directory. There the file 'Makefile' is present and by issuing the 'make' command, the code will be compiled. One can also run 'make clean' to clean up any makes performed. A 'make install' will install the compiled binaries into the bin directory. One can also issue a 'make uninstall'.

Resource management:

Within LGI it is possible to define an 'update' application capable of running perhaps a bash script on each resource to manage each resource through LGI jobs remotely. To this end the LGI_daemon reloads the configuration file and restarts logging when receiving a SIGHUP signal.

Basic command-line interface:

Running 'make' or 'make install' in the 'src' directory also makes the general command-line interface and some tools available. In the 'src' directory, apart from the 'Makefile' and some headers, the following source-files can be found:

```
binhex.cpp:         contains the algorithms to convert binary to hexadecimal strings and back.
csv.cpp:  contains the comma separated value parser for strings.
hash.cpp: contains the hash algorithm for strings.
xml.cpp:  contains the xml parser algorithms for strings.
logger.cpp:         contains the advanced logging facilities.
resource_server_api.cpp:      contains the 'project server' resource api functionality in a class.
daemon_configclass.cpp:       contains the class to read in and parse the configuration file of the daemon.
daemon_jobclass.cpp:                  contains the job directory handler class for jobs.
```

```
daemon_mainclass.cpp:                   contains the daemon main scheduling algorithms in a class.
daemon_main.cpp:              is the main daemon code that uses the above mentioned classes.
hash_main.cpp:                          a tool to hash files.
csv_main.cpp:                           a tool to extract comma separated values.
xml_main.cpp:                           a tool to parse xml tags.
hexbin_main.cpp:              a tool to convert hex to bin.
binhex_main.cpp:              a tool to convert bin to hex.
qstat_main.cpp:               a tool to list jobs on project server.
qsub_main.cpp:                          a tool to submit a job to a project server.
qdel_main.cpp:                          a tool to delete a job from a project server.
filetransfer_main.cpp:        a tool to upload, download, list and delete files in repositories.
```

All tools compiled through 'make' give details on their usage when the '-h' option is passed. The tools are mostly self-explanatory. For the utilities LGI_filetransfer, LGI_qstat, LGI_qsub and LGI_qdel, the directory ~/.LGI can be setup to hold your default configuration. These utilities also offer XML output through the '-x' option. Please study the example session below to see how the command-line interface can be setup and used. Also see the details below on repositories on how to use the LGI_filetransfer tool.

Basic web-interface:

A general and basic PHP implemented web interface can be found under the 'basic_interface' subdirectory. When a user loads a PKCS12 file containing his certificate and private key into a browser, this basic interface allows the user to look at his queue, delete jobs, submit jobs and request information on the project and the specific project server. If you do not want this interface to be accessible on your project server, adjust the Apache settings to deny access to this 'basic_interface' subdirectory (see 'SETUP.txt').

Python class interface:

In the 'python' subdirectory, through the 'LGI.py' file, the 'LGI_Client' python class is available for your python scripts. With this class you can communicate with an LGI project server from your script. This interface is available as of version 1.26 of LGI. As of version 1.35 the LGI.py is both python 2 and 3 compatible. An example script on how to use the class:

```
#!/usr/bin/python3
#
# This is a simple test code to demonstrate the use of the LGI interface class. All
# methods are being used and briefly explained. Methods that return a response of
# the project server, return a dictionary with the XML content of the response. This
# allows for easy access to parameters, as shown below. Tags transfered in binhex
# format within LGI (input and output) are automatically converted. File uploads and
# downloads from a repository are transfered streaming and the LGI_Client class can be
# be setup to run as an interface client or as a repository client as shown below.

import LGI;

# create a client instance with defaults read in from "~/.LGI"
Client = LGI.LGI_Client();

# submit a job with some input for the hello_world application at the same time upload a file
JOB = Client.SubmitJob( Application = "hello_world", Input = "haydihay", FileList = [ "/etc/resolv.conf" ] );

# extract job_id from the response dictionary
job_id = JOB[ 'LGI' ][ 'response' ][ 'job' ][ 'job_id' ];

# extract the repository url from the response dictionary
repository_url = JOB[ 'LGI' ][ 'response' ][ 'job' ][ 'job_specifics' ][ 'repository_url' ];

# setup an instance to access the repository
FileClient = LGI.LGI_Client( URL = repository_url );

# print some stuff and run some tests
print( JOB );
print();
print( Client.GetJobList( State = '!finished' ) );
print();
print( Client.GetJobState( job_id ) );
print();
print( Client.GetResourceList() );
print();
print( Client.GetServerList() );
print();

# now play with the repository for a bit
```

```
print( FileClient.ListRepository() );
FileClient.DownLoadFiles( [ "resolv.conf" ], "/tmp" );
FileClient.UpLoadFiles( [ "/etc/hosts", "/etc/fstab" ] );
print();
print( FileClient.ListRepository() );
FileClient.DeleteFiles( [ "hosts" ] );
print();
print( FileClient.ListRepository() );
print();

# finally just delete the job to clean up
print( Client.DeleteJob( job_id ) );
```

Repositories:

When a job is being submitted to LGI, the project server will automatically create a repository for the job. The repository is a special directory on the project server (or an other external storage server if desired) in which files can be uploaded to and downloaded from. Applications running on resources can now use the repository to download files from the repository and upload results to it. They should use the LGI_filetransfer utility for this.

The actual location of the repository is specified in the 'jobs_specifics' field in the database per job. It will contain two tags: “<repository> ... </repository>” and “<repository_url> ... </repository_url>”. The first tag specifies where the project server created the repository directory: “<repository> LGI@lgi.tc.lic.leidenuniv.nl:/var/www/html/LGI/repository/e3d/JOB_000f61825ff3f59c736d737612e25e3d </repository>”. The second tag shows where an interface or the basic interface can download or browse the data; “<repository_url> https://lgi.tc.lic.leidenuniv.nl//LGI/repository/e3d/JOB_000f61825ff3f59c736d737612e25e3d </repository_url>”. It is the later URL for which the LGI_filetransfer utility was written.

With the LGI_filetransfer utility files can be listed, downloaded, uploaded and deleted from a repository:

```
./LGI_filetransfer [options] command repository-url [files]

commands:

list                          list files present in specified repository.
download                      download files from repository.
upload                        upload files to repository.
delete                        delete files from repository.

options:

-h                            show this help.
-x                            output lists in XML format.
-c directory                  specify the configuration directory to read. default is ~/.LGI. specify options below
to overrule.
-j jobdirectory               specify job directory to use. if not specified try current directory or specify the
following options.
-W                            be less strickt in hostname checks of project server certificates.
-K keyfile                    specify key file.
-C certificatefile           specify certificate file.
-CA cacertificatefile        specify ca certificate file.
```

If an external repository server is desired, a project manager should make sure that on his repository server the user 'repo' exists and that project servers are allowed to use scp to transfer files in and out of the repository server without a password. This can be accomplished by adding each project server's public ssh key into the ~/.ssh/authorized_keys file of the 'repo' user on the external repository server. The public ssh key can be extracted from the x509 certificate and private key that was created for the server by the project administrator: “ssh-keygen -y -f LGI@lgi.tc.lic.leidenuniv.nl.key”.

A project server should also be setup to handle repositories. This is done through the 'inc/Config.inc' where a project server is configured. Details on how to configure a (slave) project server are given in the 'SETUP.txt' file.

On the repository server Apache should be configured to serve the repository directory and only accept clients with correctly signed certificates. How to setup and secure Apache to use certificates is also detailed upon in 'SETUP.txt'. The default sub-directory 'respository' already contains the needed CGI scripts and the .htaccess files to correctly configure Apache. This setup is ready to be used for a local repository on the project server itself.

The python class interface can also interact with a repository as is demonstrated above.

Keep in mind, when using a EXT3 file system for repositories, the limit on the number of directories is 32k. Use another file system or several project servers if you need more entries. As of version 1.30 the project server repositories support a fan-out directory layer of one level deep. This means that all jobs will have their repository directories mapped into one of 4096 subdirectories. This increases the limit of jobs on EXT3 repository file systems to more than 134 million. XFS has no such limitations. The LGI_filetransfer utility communicates with a repository server using GET, POST, PUT and DELETE requests. The PUT request, through the Apache CGI configuration and the put.cgi perl script, allows for uploads of files. The DELETE request, again through Apaches CGI configuration and the delete.cgi perl script allows for removal of file from the repository. During uploads or deletes the ACLs are checked through the '.LGI_repository_write_access' file present in each repository directory. This file contains a comma separated list of identities allowed to upload into the repository or delete files from it. These identities are matched to the credentials present in the x509 certificate used. When POSTing the 'repository' field to the repository_content.php script an XML formated list of files present in the repository is returned wrapped into "<repository_content> </repository_content>" tags as shown in the example responses of some of the interface and resource API calls above. No extra ACLs are checked for listing a repository, however, the precise (hard to guess) repository name is only know to users allowed to read details of a job. When downloading from a repository a normal GET request is used on the file to be downloaded. The Apache configuration ensures valid certificates are being used by the client.

Project management:

As of version 1.22 of LGI a special database management tool has been added in the 'tools' subdirectory:

```
ManageDB {list|add|del} {users|groups|resources} {allowed|denied} [DB [HST
    [USR [PWD]]]]
```

With this tool users, groups and resources can be easily added or deleted into the project's database. At the same time, each change to the database is recorded as an 'update' for the project into it's database. The tool is easily configured by editing the first few lines and changing the default settings. The tool should only be used at one instance of time and on the project master server only. Some examples:

```
ManageDB add users allowed exampleprojectname localhost examplemysqluser examplemysqluserpasswd

  Enter user name: theusername
  Enter application: hello_world
  Enter job limit: 10
  Enter servers to update: any

ManageDB add resources allowed

  Enter resource name: user@resource
  Enter resource url: user@resource
  Enter certificate file: resource.crt
  Enter project server flag: 0
  Enter servers to update: any

ManageDB add groups allowed
```

```
  Enter group name: newgroup
  Enter application: any
  Enter job limit: -10
  Enter servers to update: apache@exampleslaveserver.somewhere.org

ManageDB add users denied

  Enter user name: baduser
  Enter application: any
  Enter servers to update: apache@exampleserver.somewhere.org

ManageDB list users allowed

  user_name       application     job_limit
  any             hello_world     2
```

For managing your project's certificates one could use TinyCA2 (from the EPEL repository).

Scheduler:

As of version 1.25 of LGI a scheduler is implemented on the project server. The scheduler is event driven by the 'event_queue' table. This table lists the events scheduled as was described above. Currently the scheduler is used to perform some project tasks for the repository, to check if 'running' jobs are still actively monitored by their daemons, to see if lingering daemon sessions are present and to check for updates between the project servers. Future versions of the scheduler can implement an advanced quality of service for jobs by altering the 'priority' and the 'target_resources' fields of the jobs. To this end, any project server API routine that alters the 'job_queue' table schedules the 'schedule_cycle' default event and several instances of the scheduler are allowed to run concurrently. The default scheduler is based of 'first-in-first' for jobs together with 'first-come-first-serve' for resources. This means resources, when able, will poll for jobs in 'first-in-first' order. However, extra care has been taken to avoid users to claim all resources for a lengthy period of time. Please study the 'scheduler/scheduler.php' code for more details.

Storage within LGI:

Within LGI storage is implemented, in theory, through for instance a 'file_storage' application. A resource can implement such an application as desired by using the correct scripts. The idea is that jobs storing data are submitted to LGI, any resource capable of storing data accepts the job and retrieves the data to be stored through the repository of the job. The resource scripts can now store the data internally in any way they like, perhaps also remove the files from the job repository, keep a reference to the project server, 'repository_url' and 'job_id' and offer a reference to the data storage location through the job 'output' field. The job is now 'finished' as the data has been stored (this also ensures that the daemon will not monitor the job anymore and no processes are spawned for each 'running' storage job every now and then). The user can access the data through the reference in the 'output' field of the job. If the user deletes the job, the storage resource, in principle, will still retain the data. It can however, if desired, remove internal data corresponding to jobs that are not present anymore in the project server database by checking the project server for the status of the job or just by checking the existence of the 'repository_url' of the job. It could also automatically 'prune' data that has not been accessed for some time if desired as a policy. Key part is that the data stored is accessible to the user (and nobody else) through the reference put into the 'output' field of the 'file_storage' job.
All this can easily be implemented through an Apache server running an CGI interface or any other type of dynamic web-page (PHP, Python, perl...), using the x509 authentication of the LGI certificates. The reference to the data in the 'output' field can

now be a simple url to the CGI handler with an internal storage reference to the data as an url-encoded parameter. Security can be implemented by making the internal storage reference unguessable and hard to predict. The CGI handler can then offer any type of interface to that data suitable to your needs.

Currently no example is provided using the above mechanism, but another very direct storage solution using the project server repository is already possible. If a project server administrator adds the '<file_storage> </file_storage>' tags into the capabilities field of the project server in the 'active_resources' table; the LGI stack will accept jobs for the 'file_storage' application, it will create a repository on the project server for that job, and files can be uploaded, downloaded, listed and removed from that repository as long as the job is in the database. In this way the storage of files can be implemented quick-and-dirty on the project server without any extra implementations.

Scaling of LGI:

LGI has been designed from the ground-up to be scalable. Multiple project servers can be used and easily managed and the resource daemons automatically take care of this by first polling the project server they were configured with. Users can be forced to use specific project servers through the user management options of LGI too. This offers a very simple but robust (statically configurable) load balancing and scaling option. If, however, scaling or load balancing is desired without extra project servers in the LGI project, options are available and discussed below.

Extra effort was undertaken to make the LGI code scalable on all ends. The resource daemons can only run single threaded, but several daemons can run concurrently on the same resource as long as they use each a separate 'run-directory' directory. The project server PHP code was extensively profiled and the first bottleneck was found to be the number of https requests / ssl handshakes Apache can handle. This bottleneck can be vastly reduced by using keep-alive / persistent connections and by using smaller RSA keys (1024 bit). The resource daemon and interface clients provided by LGI use persistent connections as much as possible to avoid extra ssl handshakes if the project server is configured properly. The Apache configuration provided in 'SETUP.txt' has been tuned for high-performance and thus allows for persistent connections. Depending on the amount of RAM and CPU power the project server has, the number of jobs in the data base and how many resources or clients are hitting your project server concurrently, the bottleneck most likely will now be the MariaDB/MySQL back-end. Using servers with plenty RAM, cores and disks in a RAID10 configuration thus always helps.

To scale the number of web-servers to handle even more https requests / ssl handshakes on the project web-front-end (if needed), well-known techniques are available. Several Apache servers can be load-balanced over by using iptable firewalls (using a DNAT rule to a range of servers and an SNAT rule for correctly routing packages back to the originating firewall for DNAT to work), by using DNS round-robin directly to the servers or perhaps to the firewalls. All these web-servers load-balanced over, will use the same MariaDB/MySQL back-end and x509 credentials. One could also use an ssl accelerator card in the project server, but usually they are more expensive than buying extra servers. You cannot use dedicated ssl accelerators load-balancing and routing the https requests to http servers, the LGI PHP software running on the servers requires the information present in the x509 certificates.

To scale the MariaDB/MySQL back-end, care should be taken. Again, it is better to run multiple project slave servers for resource daemons to poll and users to submit to

with individual MariaDB/MySQL back-ends and storage. The resource daemons transparently take care of this. Users, perhaps depending on what interface they are using, will have jobs on different project web-front-ends unless the project administrator has implemented otherwise through the user management options of LGI. !Using (multi-master) replication, with MariaDB/MySQL proxy for instance, to scale the MariaDB/MySQL back-end is strongly discouraged! The replication introduces a delay that leads to a race-conditions in the job locking mechanisms of LGI. It is however possible to use a MariaDB/MySQL cluster back-end with a distributed database over several nodes. In this configuration, each web-front-end server (when using load-balancing across multiple web-servers) could run a local MariaDB/MySQL server, connected to a shared pool of MariaDB/MySQL storage nodes located within the same dedicated LAN. More information on this can be found at http://www.mysql.com/products/cluster. It is also possible to switch from the MyISAM and InnoDB storage engines to the HEAP or MEMORY storage engine for all tables. This will make MariaDB/MySQL faster but the database is only in RAM and not stored on disk.

Extra care has already been taken to make the use of MariaDB/MySQL as efficient as possible: tables concurrently accessed for writes use the InnoDB storage engine which implements row locking so that table locks are avoided as much as possible, tables have several indexes defined already to efficiently run the SELECT queries, no JOINS are being used, the 'job_queue' table has been normalized and indexes are used to sort the SELECTs on 'job_queue' rather than using ORDER BY. All queries have been studied with EXPLAIN and performance under a fair load (~100k jobs with ~110 concurrent resources using daemon options '-ft 5 -st 10' and testing for ~1h) was monitored with SHOW STATUS. It was found that, about 45% of the queries are UPDATEs, INSERTs or DELETEs, all table locks were immediate, 'handler_read_rnd' was 0, no queries that took 5s or more were found and less than 0.01% of the queries took more than 1s, the 'max_used_connections' was found to be 105, 'open_tables' was 196 and 'threads_connected' 92 on average. Also, on average, 'qcache_hits' was two times bigger than 'qcache_inserts' and 'qcache_not_cached' was 25% of the total number of query cache lookups. These numbers have been obtained running the suggested configuration of 'SETUP.txt' on a dual Xeon 3.0 GHz Nocona EM64T machine with 4GB of DDRII 400MHz RAM using Scientific Linux 4.3 and hyperthreading turned on. The load of that system during testing was found to be ~31 (85% user time, 15% system time) and MariaDB/MySQL was found to take about 15 to 50% of CPU load. No swapping was seen on the system and the MariaDB/MySQL disk usage was ~250MB and memory usage was about ~2.6GB (of which ~300MB was resident in RAM). During these tests 4k RSA keys were used for the x509 certificates.

Furthermore, the PHP profiling showed that for interfaces, on average 4 queries are performed per project server request (deleting jobs is the most expensive operation and takes 7 queries) and for resources on average 7 queries are performed per project server request excluding the requests for work (only 3 queries are performed when checking the status of a job by a resource, but 9 if a job lock removal is requested by a resource; the most expensive operations are locking, updating and unlocking jobs for resources). Resources requesting for work on average took 134 queries.

To scale up a repository, also plenty of well-known techniques are available. One could use GlusterFS storage using the web-front-end servers as storage bricks with local RAID10 disks or traditional SAN with NFS techniques can be used. Key is that all web-front-end servers being load balanced over in a project pool using a single MariaDB/MySQL back-end have access to the same (distributed) storage transparently. However, it is again advised to use several project slave servers with independent MariaDB/MySQL back-ends and repository storage local to each server.

Also keep in mind that the EXT3 file system has a limit of 32k directory entries. The EXT4 or XFS file systems have no such limit. As of version 1.30 the project server repositories support a fan-out directory layer of one level deep. This means that all jobs will have their repository directories mapped into one of 4096 subdirectories. This increases the limit of jobs on EXT3 repository file systems to more than 134 million.

Spec files and RPMs:

As of version 1.30 .spec files have been created to allow for easy deployment of all components of LGI on RHEL based systems. Currently RPMs can be built for x86_64 systems using RHEL (or derivatives) 7, 8 and 9. In the specs subdirectory a script can be run to compile the latest LGI software into 4 RPMs: LGI_cli, LGI_python, LGI_server and LGI_resource. These RPMs automatically configure the system into a working project LGI or resource and take care of dependencies. The RPMs can be compiled anywhere and are relocatable. They also will offer LGI_daemon and LGI_server init scripts and log rotations. Take note; the LGI_server rpm is an interactive rpm and is not designed to be installed unattended and is best installed after a 'minimal install' of a RHEL server.

Example user session:

In this part an example user session is demonstrated with the 'hello_world' application.

```
[mark@lgi ~]$ cd LGI/src
[mark@lgi src]$ make install
g++ -O2 -D_FILE_OFFSET_BITS=64 -D_LARGEFILE_SOURCE -c logger.cpp
g++ -O2 -D_FILE_OFFSET_BITS=64 -D_LARGEFILE_SOURCE -c resource_server_api.cpp
g++ -O2 -D_FILE_OFFSET_BITS=64 -D_LARGEFILE_SOURCE -c xml.cpp
g++ -O2 -D_FILE_OFFSET_BITS=64 -D_LARGEFILE_SOURCE -c csv.cpp
g++ -O2 -D_FILE_OFFSET_BITS=64 -D_LARGEFILE_SOURCE -c hash.cpp
g++ -O2 -D_FILE_OFFSET_BITS=64 -D_LARGEFILE_SOURCE -c binhex.cpp
g++ -O2 -D_FILE_OFFSET_BITS=64 -D_LARGEFILE_SOURCE -c daemon_configclass.cpp
g++ -O2 -D_FILE_OFFSET_BITS=64 -D_LARGEFILE_SOURCE -c daemon_jobclass.cpp
g++ -O2 -D_FILE_OFFSET_BITS=64 -D_LARGEFILE_SOURCE -c daemon_mainclass.cpp
g++ -O2 -D_FILE_OFFSET_BITS=64 -D_LARGEFILE_SOURCE -c daemon_main.cpp
g++ -O2 -o LGI_daemon logger.o resource_server_api.o xml.o csv.o hash.o binhex.o daemon_configclass.o
daemon_jobclass.o daemon_mainclass.o daemon_main.o -lcurl
g++ -O2 -D_FILE_OFFSET_BITS=64 -D_LARGEFILE_SOURCE -c xml_main.cpp
g++ -O2 -o xml xml.o csv.o xml_main.o
g++ -O2 -D_FILE_OFFSET_BITS=64 -D_LARGEFILE_SOURCE -c binhex_main.cpp
g++ -O2 -o binhex binhex.o binhex_main.o
g++ -O2 -D_FILE_OFFSET_BITS=64 -D_LARGEFILE_SOURCE -c hexbin_main.cpp
g++ -O2 -o hexbin binhex.o hexbin_main.o
g++ -O2 -D_FILE_OFFSET_BITS=64 -D_LARGEFILE_SOURCE -c hash_main.cpp
g++ -O2 -o hash hash.o binhex.o hash_main.o
g++ -O2 -D_FILE_OFFSET_BITS=64 -D_LARGEFILE_SOURCE -c csv_main.cpp
g++ -O2 -o csv csv.o csv_main.o
g++ -O2 -D_FILE_OFFSET_BITS=64 -D_LARGEFILE_SOURCE -c filetransfer_main.cpp
g++ -O2 -o LGI_filetransfer logger.o resource_server_api.o xml.o csv.o hash.o binhex.o daemon_configclass.o
daemon_jobclass.o filetransfer_main.o -lcurl
g++ -O2 -D_FILE_OFFSET_BITS=64 -D_LARGEFILE_SOURCE -c interface_server_api.cpp
g++ -O2 -D_FILE_OFFSET_BITS=64 -D_LARGEFILE_SOURCE -c qstat_main.cpp
g++ -O2 -o LGI_qstat logger.o interface_server_api.o xml.o csv.o binhex.o qstat_main.o -lcurl
g++ -O2 -D_FILE_OFFSET_BITS=64 -D_LARGEFILE_SOURCE -c qdel_main.cpp
g++ -O2 -o LGI_qdel logger.o interface_server_api.o xml.o csv.o binhex.o qdel_main.o -lcurl
g++ -O2 -D_FILE_OFFSET_BITS=64 -D_LARGEFILE_SOURCE -c qsub_main.cpp
g++ -O2 -o LGI_qsub logger.o interface_server_api.o resource_server_api.o daemon_jobclass.o daemon_configclass.o
xml.o csv.o binhex.o hash.o qsub_main.o -lcurl
mkdir -p ../bin; cp .htaccess index.html LGI_daemon xml binhex hexbin hash csv LGI_filetransfer LGI_qstat LGI_qsub
LGI_qdel ../bin
rm -rf ../daemon/bin; ln -s ../bin ../daemon/bin
[mark@lgi src]$ ./LGI_
LGI_daemon        LGI_filetransfer  LGI_qdel          LGI_qstat         LGI_qsub
[mark@lgi src]$ ./LGI_qstat

-------------------------------------------------------------------------------------------------------------------
---------------
    # | job_id |        state |                    target_resources |      application |               time_stamp
|       owners
-------------------------------------------------------------------------------------------------------------------
---------------
-------------------------------------------------------------------------------------------------------------------
---------------

Number of jobs listed : 0
This project          : LGI
This project server   : https://lgi.tc.lic.leidenuniv.nl/LGI
Project master server : https://lgi.tc.lic.leidenuniv.nl/LGI

[mark@lgi src]$ ./LGI_qsub -a hello_world -t any

Job has been submitted. Some details:
```

```
This project          : LGI
This project server   : https://lgi.tc.lic.leidenuniv.nl/LGI
Project master server : https://lgi.tc.lic.leidenuniv.nl/LGI
User                  : mark@laptop
Groups                : theor
Job id                : 5878300
Job state             : queued
Job application       : hello_world
Job specifics         : <default>hello</default> <repository>
LGI@lgi.tc.lic.leidenuniv.nl:/var/www/html/LGI/repository/454/JOB_9750a4a27fb47ad48f9f23f213d66454 </repository>
<repository_url> https://lgi.tc.lic.leidenuniv.nl/LGI/repository/454/JOB_9750a4a27fb47ad48f9f23f213d66454
</repository_url>
Target resources      : any
Job owners            : mark@laptop, theor
Read access on job    : admin, mark@laptop
Write access on job   : admin, mark@laptop
Time stamp            : Wed Jul 10 11:30:49 2019 [1562751049]
Input                 :
Repository content    : .LGI_repository_write_access

[mark@lgi src]$ LGI_qstat

-------------------------------------------------------------------------------------------------------------------------------
-----------------
    # | job_id |        state |                    target_resources |      application |               time_stamp
|       owners
-------------------------------------------------------------------------------------------------------------------------------
----------------
    1 | 5878300 |      running |            mark@boeddha.gorlaeus.net |      hello_world | Wed Jul 10 11:30:53 2019
| mark@laptop, theor
-------------------------------------------------------------------------------------------------------------------------------
----------------

Number of jobs listed : 1
This project          : LGI
This project server   : https://lgi.tc.lic.leidenuniv.nl/LGI
Project master server : https://lgi.tc.lic.leidenuniv.nl/LGI

[mark@lgi src]$ LGI_qstat 5878300

This project          : LGI
This project server   : https://lgi.tc.lic.leidenuniv.nl/LGI
Project master server : https://lgi.tc.lic.leidenuniv.nl/LGI
User                  : mark@laptop
Groups                : theor
Job id                : 5878300
Job state             : running
Job application       : hello_world
Job specifics         : <daemon_reference> 5A7FD65D608BAD2C9AB217BBE01F87CC </daemon_reference>
<default>hello</default> <repository>
LGI@lgi.tc.lic.leidenuniv.nl:/var/www/html/LGI/repository/454/JOB_9750a4a27fb47ad48f9f23f213d66454 </repository>
<repository_url> https://lgi.tc.lic.leidenuniv.nl/LGI/repository/454/JOB_9750a4a27fb47ad48f9f23f213d66454
</repository_url>
Target resources      : mark@boeddha.gorlaeus.net
Job owners            : mark@laptop, theor
Write access on job   : admin, mark@laptop
Read access on job    : admin, mark@laptop
Time stamp            : Wed Jul 10 11:30:53 2019 [1562751053]
Input                 :
Output                :
Repository content    : .LGI_repository_write_access

[mark@lgi src]$ LGI_filetransfer list
https://lgi.tc.lic.leidenuniv.nl/LGI/repository/454/JOB_9750a4a27fb47ad48f9f23f213d66454

.LGI_repository_write_access           18 Wed Jul 10 11:30:49 2019 [1562751049]

[mark@lgi src]$ LGI_filetransfer upload
https://lgi.tc.lic.leidenuniv.nl/LGI/repository/454/JOB_9750a4a27fb47ad48f9f23f213d66454 ./binhex.h

Uploaded to 'https://lgi.tc.lic.leidenuniv.nl/LGI/repository/454/JOB_9750a4a27fb47ad48f9f23f213d66454/binhex.h' ...

[mark@lgi src]$ LGI_qstat

-------------------------------------------------------------------------------------------------------------------------------
-----------------
    # | job_id |        state |                    target_resources |      application |               time_stamp
|       owners
-------------------------------------------------------------------------------------------------------------------------------
----------------
    1 | 5878300 |      running |            mark@boeddha.gorlaeus.net |      hello_world | Wed Jul 10 11:30:53 2019
| mark@laptop, theor
-------------------------------------------------------------------------------------------------------------------------------
----------------

Number of jobs listed : 1
This project          : LGI
This project server   : https://lgi.tc.lic.leidenuniv.nl/LGI
Project master server : https://lgi.tc.lic.leidenuniv.nl/LGI

[mark@lgi src]$ LGI_qstat 5878300

This project          : LGI
This project server   : https://lgi.tc.lic.leidenuniv.nl/LGI
Project master server : https://lgi.tc.lic.leidenuniv.nl/LGI
User                  : mark@laptop
Groups                : theor
Job id                : 5878300
Job state             : running
Job application       : hello_world
Job specifics         : <daemon_reference> 5A7FD65D608BAD2C9AB217BBE01F87CC </daemon_reference>
<default>hello</default> <repository>
LGI@lgi.tc.lic.leidenuniv.nl:/var/www/html/LGI/repository/454/JOB_9750a4a27fb47ad48f9f23f213d66454 </repository>
<repository_url> https://lgi.tc.lic.leidenuniv.nl/LGI/repository/454/JOB_9750a4a27fb47ad48f9f23f213d66454
</repository_url>
Target resources      : mark@boeddha.gorlaeus.net
Job owners            : mark@laptop, theor
Write access on job   : admin, mark@laptop
Read access on job    : admin, mark@laptop
Time stamp            : Wed Jul 10 11:30:53 2019 [1562751053]
Input                 :
Output                :
```

```
Repository content    : binhex.h, .LGI_repository_write_access

[mark@lgi src]$ LGI_filetransfer list
https://lgi.tc.lic.leidenuniv.nl/LGI/repository/454/JOB_9750a4a27fb47ad48f9f23f213d66454

binhex.h                             1257 Wed Jul 10 11:31:36 2019 [1562751096]
.LGI_repository_write_access           18 Wed Jul 10 11:30:49 2019 [1562751049]

[mark@lgi src]$ LGI_filetransfer download
https://lgi.tc.lic.leidenuniv.nl/LGI/repository/454/JOB_9750a4a27fb47ad48f9f23f213d66454 /tmp/binhex.h

Downloaded from 'https://lgi.tc.lic.leidenuniv.nl/LGI/repository/454/JOB_9750a4a27fb47ad48f9f23f213d66454/binhex.h'
...

[mark@lgi src]$ ls -laFh /tmp/binhex.h
-rw-rw-r--. 1 mark mark 1.3K Jul 10 11:32 /tmp/binhex.h
[mark@lgi src]$ LGI_qstat

--------------------------------------------------------------------------------------------------------------------
----------------
    # | job_id |        state |                    target_resources |      application |               time_stamp
|       owners
--------------------------------------------------------------------------------------------------------------------
----------------
    1 | 5878300 |      running |            mark@boeddha.gorlaeus.net |      hello_world | Wed Jul 10 11:30:53 2019
| mark@laptop, theor
--------------------------------------------------------------------------------------------------------------------
----------------

Number of jobs listed : 1
This project          : LGI
This project server   : https://lgi.tc.lic.leidenuniv.nl/LGI
Project master server : https://lgi.tc.lic.leidenuniv.nl/LGI

[mark@lgi src]$ ls -laFh ~/.LGI
total 64K
drwxrwxr-x. 2 mark mark 4.0K Jan  7  2019 ./
drwx------. 9 mark mark 4.0K Jul  9 09:46 ../
-rw-rw-r--. 1 mark mark 4.3K Nov 14  2017 ca_chain
lrwxrwxrwx. 1 mark mark   15 Jan 16  2017 certificate -> mark@laptop.crt
-rw-rw-r--. 1 mark mark   12 Aug 17  2015 defaultapplication
-rw-rw-r--. 1 mark mark   25 Aug 17  2015 defaultjobspecifics
-rw-------. 1 mark mark    4 Aug 17  2015 defaultproject
-rw-------. 1 mark mark   37 May  5  2016 defaultserver
-rw-rw-r--. 1 mark mark    4 May  5  2016 defaulttargetresources
-rw-------. 1 mark mark    6 Aug 17  2015 groups
-rw-------. 1 mark mark 2.1K Jan  7  2019 mark@laptop.crt
-rw-------. 1 mark mark 1.8K Feb 10  2016 mark@laptop.csr
-rw-------. 1 mark mark 3.2K Jan 16  2018 mark@laptop.key
-rw-------. 1 mark mark 7.4K Jan  7  2019 mark@laptop.pfx
lrwxrwxrwx. 1 mark mark   15 Jan 16  2017 privatekey -> mark@laptop.key
-rw-------. 1 mark mark   12 Aug 17  2015 user
[mark@lgi src]$ LGI_qdel 5878300

Aborting job 5878300 in project LGI on server https://lgi.tc.lic.leidenuniv.nl/LGI

[mark@lgi src]$ LGI_qstat

--------------------------------------------------------------------------------------------------------------------
----------------
    # | job_id |        state |                    target_resources |      application |               time_stamp
|       owners
--------------------------------------------------------------------------------------------------------------------
----------------
    1 | 5878300 |     aborting |            mark@boeddha.gorlaeus.net |      hello_world | Wed Jul 10 11:35:00 2019
| mark@laptop, theor
--------------------------------------------------------------------------------------------------------------------
----------------

Number of jobs listed : 1
This project          : LGI
This project server   : https://lgi.tc.lic.leidenuniv.nl/LGI
Project master server : https://lgi.tc.lic.leidenuniv.nl/LGI

[mark@lgi src]$_
```