GRMS User Guide

Gridge 1.0

Piotr Kopta <pkopta@man.poznan.pl>
Krzysztof Kurowski <krzysztof.kurowski@man.poznan.pl>
Bogdan Ludwiczak <bogdanl@man.poznan.pl>
Ariel Oleksiak <ariel@man.poznan.pl>
Tomasz Piontek <piontek@man.poznan.pl>
Juliusz Pukacki <pukacki@man.poznan.pl>

GRMS User Guide: Gridge 1.0

by Piotr Kopta, Krzysztof Kurowski, Bogdan Ludwiczak, Ariel Oleksiak, Tomasz Piontek, and Juliusz Pukacki

This guide is part of GRIDGE documentation

Table of Contents

List of Examples

Chapter 1. Introduction

The Gridge Resource Management System (GRMS) is an open source meta-scheduling system, which allows developers to build and deploy resource management systems for large scale distributed computing infrastructures. GRMS, based on dynamic resource selection, mapping and advanced scheduling methodology, combined with feedback control architecture, deals with dynamic Grid environment and resource management challenges, e.g. load-balancing among clusters, remote job control or file staging support. Therefore, the main goal of GRMS is to manage the whole process of remote job submission to various resources such as batch queuing systems or single computational nodes. Finally, GRMS can be considered as a robust system which provides abstraction of the complex Grid infrastructure as well as a toolbox which helps to form and adapts to distributing computing environments. GRMS has been designed as an independent set of components for resource management processes which can take advantage of various low-level Core Services, e.g. GRAM, GridFTP and Gridge Monitoring System, as well as various Grid middleware services, e.g. Gridge Authorization Service (GAS), Gridge Data Management Service (DMS). All these services working together provide a consistent, adaptive and robust Grid middleware layer which fits dynamically to many different distributing computing infrastructures. The GRMS implementation requires Globus software to be installed on Grid resources, and uses Globus Core Services deployed on resources: GRAM, GridFtp, MDS (optional). GRMS supports Grid Security Infrastructure by providing the GSI-enabled web service interface for all clients, e.g. portals or applications, and thus can be integrated with any other middleware Grid environment. One of the main assumptions for GRMS is to perform remote jobs control and management in the way that satisfies Users (Job Owners) and their applications requirements. All users requirements are expressed within XML-based resource specification documents and sent to GRMS as SOAP requests over secure connections. Simultaneously, Resource Administrators (Resource Owners) have full control over resources on which all jobs and operations will be performed by appropriate GRMS setup and installation. Note, that GRMS together with Core Services reduces operational and integration costs for Administrators by enabling Grid deployment across previously incompatible cluster and resources. Technically speaking, GRMS is a persistent service within a Tomcat/Axis container. It is written completely in Java so it can be deployed on various platforms. With the Gridge Authorization Service, GRMS is able to manage both, job grouping and jobs within collaborative environments according to predefined VO security rules and policies. With the Data Management services from Gridge, GRMS can create and move logical files/ catalogs and deal with data intensive experiments. Gridge Monitoring Service can be used by GRMS as an additional information system. Finally, Mobile service can be used to send notifications via SMS/emails about events related to users' jobs and as a gateway for GRMS mobile clients. GRMS is able to store all operations in a database. Based on this information a set of very useful statistics for both end users and administrators can be produced. All the data are also a source for further, more advanced analysis and reporting tools.

Chapter 2. GRMS Functionality

GRMS offers complete set of capabilities to serve resource management in Grid environments. The most important features of GRMS are:

- Job and task submission
- Job control (cancel, suspend, resume)
- Choosing the “best” resource for the Job execution, according to Job Description and chosen mapping algorithm – multicriteria algorithm
- Submitting the GRMS Job according to provided Job Description to chosen resource
- Job migration
- Registering an application callback information
- Application checkpointing:
    - using defined checkpoint interface implemented by application
- Complex information about submitted jobs and tasks
    - List of jobs submitted by user
    - Information about the Job status
    - Job Description used for submission
    - Information about request progress
    - Name of host where the Job is running
    - Submission time
    - Start time on resource
    - Finish time
    - History of job execution (migrations)
- Dynamic resource discovery
- Ability to use multiple information sources about Grid environment (standard Globus MDS (GIIS/GRIS infrastructure), iGrid, Delphoi, Mercury)
- Support for file staging – transferring input and output files and whole directories (GridFTP, GASS, Gridge Data Management System, SRB, RFT)
- Mechanism for registering for events notification
- Notifying about status changes and request progress (e.g. via e-mail or sms)
- Time constraints for running jobs
- Dynamically extending (during application runtime) job description by adding output data
- Support for grouping jobs in projects (notifications for project)
- Support for workflow jobs: job can consist of set of independent tasks with or without precedence constraints

Chapter 3. GRMS API

GRMS API provided for end users, can be divided into several groups. The following groups can be distinguished:

- Job submission and control,
- Task submission and control,
- Listing jobs according to specified criteria,
- Listing tasks belonging to the given job according to specified criteria,
- Managing tasks,
- Getting information about jobs,
- Getting information about tasks,
- Getting a list of resources that meet user's requirements and criteria,
- Managing notifications,
- Auxiliary functionality.

Next sections describe all these functionality groups.

Job submission and control

This group of functionality allows to submit and control whole jobs. GRMS treats jobs as a sets of dependent tasks that constitute a logical whole. Each task can be executed by GRMS only if all tasks it depends on are in specified states. The whole job is described by XML document called "job description" containing information about the job as a whole (job properties) and information about all its tasks needed to map tasks to resources and to execute them in a proper way (dependencies between tasks, locations of executables, files that have to be staged in and/or out, arguments, environment variables, checkpoint files, etc.).

The following functionality can be used for job submission and control:

- `submitJob` – it is the main functionality of GRMS. Using it a user can submit a job (a set of tasks) described by the GRMS Job Description to be executed by GRMS. If the description is valid GRMS returns to the user a globally unique job identifier (GRMS_JOB_ID), which unambiguously identifies the job in the system
- `commitJob` - this functionality allows to approve for processing a job submitted with two phase commit mechanism. The two phase commit mechanism can be used to register notifications before processing of the job will be started by GRMS.
- suspendJob – using this functionality user is able to suspend the running job (all running tasks). It means that each running task forming part of job will be checkpointed and all checkpoint files/directories will be staged out. If a new job description is not defined the previous one will be used.
- `resumeJob` – this functionality resumes the execution of the job which was previously suspended. It is possible to define a new job description or to use the previous one.
- `cancelJob` – this functionality allows the user to stop the running job. The difference between the `suspendJob` and `cancelJob` functionalities is that in case of "`cancelJob`" funcionality all tasks are stopped (killed) by the system and checkpoint operation is not performed for running tasks.
- `recoverJob` - because experiments (jobs) controlled by GRMS can consist of huge amount of potentially time-consuming tasks that proper execution depends not only on the correct job description, but can be also

broken due to some unpredictable event. The `recoverJob` functionality allows to restart the job, skipping execution of previously finished tasks, taking their results produced previously.

- `refreshJobProxy` - for job submission and then execution of tasks on resources GRMS uses the time-limited user proxy, which can expire during the processing of job causing lost of control on running tasks and making impossible to start new ones. This can happen very often specially for long running jobs, which finish time is hard to predict and strongly related to available resources. Addressing the issue GRMS allows to prolong the user proxy for the whole job using `refreshJobProxy` functionality.

Task submission and control

This functionality has similar meaning to the corresponding one described in previous section. The difference is that it concerns not the whole job but single tasks. The following functionality can be used for task submission and control:

- `migrateTask` – this functionality allows the user to migrate one of tasks to a " better" resource (if such one exists) to improve task performance or system utilization. The task is identified by the job identifier returned as a result of job submission and task identifier specified by the user in job description. To be migrateable task has to be checkpointable. When application is checkpointable on demand (implements relatively simple "`checkpoint`" web service interface, which location is registered in GRMS. In this case the whole process of task migration is relatively simple. The task to be migrated is checkpointed on the resource, which it is currently running on and then restarted on a new one pointed by the user or chosen by GRMS. GRMS is able to migrate also applications not implementing the aforementioned interface. In this case application periodically has to perform checkpoint procedure saving files to any storage and during the migration process it is just killed by GRMS and then the execution is resumed from the state saved in the last checkpoint file. Both described above cases are typical examples of application/user-level checkpointing, requiring from the application developer to implement mechanisms for storing application data to checkpoint files, and assumes that checkpointing procedure is hard coded in the application. The migration process can be performed by GRMS according to a new job description passed as the parameter. If the new job description was not defined GRMS tries to perform the request basing on the job description passed during the submission or the previous migration request. The migration is done only when a better resource was found.

- `commitTask` - this functionality allows to approve task submitted as demanding commitment for processing by GRMS. Processing of every task, irrespective of precedence constraints resulting from dependencies between tasks, can be postponed until it will be approved by commitment (two phase commit mechanism).

- `suspendTask` – using this functionality the user is able to suspend running task. It means that the task forming part of job will be checkpointed and all checkpoint files/directories will be staged out. If a new job description is not defined the previous one will be used.

- `resumeTask` – this functionality resumes the execution of the previously suspended task. It is possible to define a new job description or to use the previous one.

- `cancelTask` – allows to stop (kill) the running task. The difference between the `suspendTask` and `cancelTask` functionalities is that the task to be cancelled is stopped (killed) by the system and the checkpoint operation is not performed.

- `extendTaskExecutionTime` - addressing many advanced scenarios GRMS is able to handle tasks having some time constrains and requirements, like for example specified period of time when the execution of task must start or the duration of task execution. The `extendTaskExecutionTime` functionality allows to prolong the execution of time scheduled tasks.

Listing jobs according to specified criteria

GRMS is able to return a list of jobs (their identifiers) belonging to the user that invoked the request or to a specified project. It is possible to query for all jobs or a subset of jobs in specific state. The following operations provide the aforementioned functionality:

- `getJobsList` – returns a list of jobs belonging to the user, optionally it is possible to define the requested status,
- `getAllJobsList` – returns a list of all jobs in given state.
- `getProjectJobsList` - returns a list of jobs belonging to the specified project, optionally it is possible to define the requested status,

Every job is in the system must be in one of the following states.

- `UNCOMMITTED` - the job was submitted with two phase commit option and waits to be committed,
- `SUBMITTED` – the job was submitted to the system and waits for the execution,
- `SUSPENDED` – the job was suspended,
- `ACTIVE` – the job is active,
- `FINISHED` – the job was completed,
- `FAILED` – the job (at least one crucial task belonging to the job) failed, there could be many reasons of this (for example, GRMS was not able to find the requested resource or copy all needed files),
- `CANCELED` – the job was canceled by the user,
- `BROKEN` - one or more of crucial tasks failed, GRMS waits until active tasks will finish and change the status of the job to FAILED.

Listing tasks according to specified criteria

GRMS is able to return a list of tasks (identifiers) being a part of concrete job. It is possible to query for all tasks or subset of all tasks in given state. The following method provides the aforementioned functionality:

- `getTasksList` – returns a list of tasks, optionally it is possible to define the requested status,

Below, there is a full list of task statuses in the system:

- `UNSUBMITTED` – the task cannot be started because of dependencies,
- `UNCOMMITED` - the task waits to be commited,
- `QUEUED` – the task was put into the queue and waits for execution,
- `PREPROCESSING` – GRMS makes some actions needed to start the task (looks for the resource, staging in files),
- `PENDING` – the task is pending in the queueing-system,
- `RUNNING` – the task is active,
- `STOPPED` – the task was finished or was checkpointed, but GRMS did not start staging out files,
- `POSTPROCESSING` – GRMS makes some actions needed to complete the task, for example staging out files, clearing working environment, etc.,
- `FINISHED` – the task was completed,
- `SUSPENDED` – the task was suspended,

- `FAILED` – the task failed, there could be many reasons of this (for example, GRMS was not able to find the requested resource or copy all needed files),
- `CANCELED` – the task was canceled by the user.

Managing tasks

This functionality gives the user possibility to register and unregister the task in GRMS for checkpointing and manage these settings.

Moreover, the user is able to manage dynamically output, checkpoint files and directories.

- `registerTaskApplicationAccess` – This functionality allows to register information needed for checkpointing the task. To be checkpointable the task has to register itself in GRMS passing to the system its GRMS_JOB_ID and GRMS_TASK_ID (both can be taken from the environment variables set up by GRMS during the submission process) and the address of the Web Service implementing the “checkpoint API”. Using this interface GRMS is able to send the checkpoint request to the application and triggers off the checkpointing. This is example of application-level checkpointing, which requires the application developer to implement mechanisms for storing all data to a checkpoint file. In other words, checkpointing is hard coded in the application. This kind of checkpointing is obviously much more portable and more applicable in Grids. Note that the application developer has to implement all internal mechanisms to write a checkpoint file to the local disk when the application receives the checkpoint call from GRMS. If the task has no registered information about location of "checkpoint" web service interface GRMS performing the "migrate" request tries to checkpoint the application using periodical checkpoint if such one exists.
- `unregisterTaskApplicationAccess` - this functionality gives the possibility to unregister the job for checkpointing,
- `getTaskApplicationAccess` - returns information about settings related to the "checkpoint" functionality,
- `addTaskFileDirs` – this functionality allows to register dynamically additional output/checkpoint files and/or directories for the task with given id. The location of file/directory can be expressed as a physical or logical path.
- `getTaskFileDirs` – returns a list of output/checkpoint files and directories registered for the given task. It is possible to filter files according to logical type (inputs, outputs, checkpoints) and their origin (job description or dynamic registration),
- `deleteTaskFileDirs` – allows to unregister the set of output/checkpoint files and/or directories.

Getting information about the job

This functionality gives the user a possibility to get complex information about the job with a given identifier.

- `getJobInformation` – returns the general information about the job
  - *`project`* – project, which the job belongs to,
  - *`userDn`* – user Distinguish Name,
  - *`status`* – status of the job,
  - *`waitForCommit`* - indicates if the job waits for commit,
  - *`submissionTime`* – submission time,
  - *`finishTime`* – finish time or null if the time is unknown,
  - *`errorDescription`* – message describing the cause of last error,

- *taskIdentifiers* - list of tasks forming the job,
- *taskCount* - number of tasks,
- *jobDescription* – description of the job.

Getting information about the task

This functionality gives the user a possibility to get complex information about the task with a given identifier.

- getTaskInformation – this functionality returns the general information about the task
  - *type* - type of the task (single, multiple, mpi, mpichg),
  - *status* – status of the task,
  - *waitForCommit* - indicates if the task waits for commit,
  - *submissionTime* – submission time,
  - *finishTime* – finish time or null if the time is unknown,
  - *proxyLifetime* - lifetime of the proxy associated with this task,
  - *requestStatus* - status of GRMS request,
  - *errorDescription* – message describing the cause of last error,
  - *historyLength* - length of the history of the task,
  - *history* - list of items describing history of the task's life. Every item of this history contains the following information:
    - *startTime* – time when GRMS started processing th task,
    - *localSubmissionTime* – time when the task was submitted on the local resource,
    - *localStartTime* – time when the task was started on the local resource
    - *localFinishTime* – time when the task was finished on the local resource,
    - *taskDescription* - description of the task - part of job description concerning the given task,
    - *applicationAccess* - location of the service that can be used to checkpoint the application and the process identifier of application.
    - *coallocation* - list of coallocation items. Every item contains information about one host:
      - *hostName* - name of the host,
      - *count* - number of processes running on this host,
      - *indexes* - mpichg indexes of processes running on this host.

Finding resources

GRMS is able to find a list of resources that meet user requirements expressed in the job description in the “resources” sections (for example, parameters connected with operating system – name, version, release, name of host, local resource management system, minimal amount of memory in MB, minimal number of processors,

minimal speed of cpu(s), etc.).

- `findResources` - returns a list of resources (in the form of resource manager contact strings) that meet resource requirements from the job description. Because job can consist of many tasks user has to specify task, which the system should find resources for.

Notifications

GRMS also provides support for events notifications. A notification mechanism is very general and designed to allow clients to receive information in asynchronous way. User can register for notifications concerning the whole job or single task. The difference except the obvious one is that in case of tasks GRMS is able to send to the registered clients two kinds of notifications: the “status notification” connected with changes concerning a life cycle of the task and the “request notification” related to the performed GRMS request. Jobs have only "status notifications". Currently GRMS is able to send notifications in two ways: using SOAP protocol and writing to a remote file. Registered notifications can be queried according to some criteria and unregistered. For every registered notification GRMS is able to return detailed information describing it.

The following set of functionalities can be used to manage GRMS notifications:

- `registerJobStatusNotification` - this functionality allows to register location of service or remote file for notifications concerning changes of job status. GRMS is able to send notification every time when the status of job has changed (for example from `ACTIVE` to `FINISHED` or `FAILED`). It is possible to register only for some subset of statuses.
- `getJobStatusNotifications` - this functionality lists all notifications concerning given job,
- `getJobStatusNotification` - this functionality returns information concerning notification with given id,
- `unregisterJobStatusNotification` - allows the user to unregister the notification with given id,

- `registerTaskStatusNotification` - has the same meaning as registerJobStatusNotification, but concerns a life cycle of a single task,
- `registerTasksStatusNotification` - registers in one call notifications for all tasks forming a job,
- `registerTaskRequestStatusNotification` - registers for given task notification concerning processing of request,
- `getTasksNotifications` - lists descriptions of notifications registered for given task. It is possible to specify type of event (status vs. request).
- `getTaskNotification` - for a given task returns information concerning notification with given id,
- `getTaskStatusNotification` - for a given task returns information concerning "status"-notification with given id,
- `getTaskRequestStatusNotification` - for a given task returns information concerning "request"-notification with given id,
- `getTaskStatusNotifications` - returns list of "status"-notifications registered for a given task,
- `getTaskRequestStatusNotifications` - returns list of "request"-notifications registered for a given task,
- `unregisterTaskNotification` - allows the user to unregister the notification with given id.

Every notification sent by grms contains following information:

- *jobId* - GRMS job identifier [%j],
- *notificationId* - notification identifier, [%n]
- *project* - project which the job or task belongs to, [%p]
- *time* - time when the event occurred, t is possible to specify if the time should be in human readable format "January 25, 2005, 17:31:10 GMT" [%c] or as a number of milliseconds since January 1, 1970, 00:00:00 GMT [%C]
- *user* - user whom the job or task belongs to, [%u]
- *registrator* - user who registered for this notification, [%r]
- *errorDescription* - message describing the cause of last error, [%d]

Depending on type of notification it can contain also:

- *taskId* - task identifier, [%t]
- *status* - job or task status. In case of task the status can be life cycle or request one. [%s]

Important

A letter in square brackets after the description of parameter is a mark that represents following information in string describing format of the notification message for GASS notifications. The format string can contains any text in which set of defined above marks will be replaced by information taken from GrmsNotification. For example "Job %j has changed it status to %s at %c !".

Auxiliary functionality

The functionality listed below has no productive character, but can be useful for testing purposes.

- testJobDescription - GRMS gives the possibility to check the correctness of the job description using the “testJobDescription” functionality. In the case of incorrectness of description GRMS returns the diagnostic information describing the syntax error.
- getServiceDescription – this functionality allows to get a description of GRMS Web Service interface. It is possible to get the description in a short or in a full version. The first one is limited only to the name of the service and its version. The second one contains additionally some diagnostic information (locations of service and client, user’s Distinguish Name) and the detailed description of Web Service interface.

Chapter 4. GRMS Job Description (GJD)

As it was discussed in the previous section all information needed to execute a job including resource requirements have to be described in the form of the GRMS Job Description. It is an XML-based document, which allows users to specify a description of a job executable, job resource requirements, needed data, dependencies between tasks etc. Generally, the following parameters are available in the GRMS Job Description:

- tasks executables:
  - location of executables,
  - arguments,
  - file arguments (files that have to be accessible in a working directory of the running executable),
  - environment variables,
  - standard input,
  - standard output,
  - standard error,
  - checkpoint definition.
- resource requirements:
  - name of host for the task execution (if provided no scheduling algorithm is used),
  - operating system,
  - required local resource management system (lsf, pbs, condor, etc.),
  - minimum memory required,
  - minimum number of CPUs required,
  - minimum speed of the CPU,
  - required applications installed at destination hosts,
  - network parameters (bandwidth, latency and capacity).
- constraints:
  - hard constraints,
  - soft constraints
- tasks execution times:
  - task execution time,
  - time slot (e.g. from 10.00 till 16.00),
  - time period (e.g. till 31st March except Sundays).
- workflow:

- dependencies between tasks.

Locations of files can be specified as gridFTP/GASS urls as well as logical ones using a data management system that supports logical file names.

GJD Specification

<grmsJob> GRMS Job Description starts with the <grmsJob> element, which contains the "appid" attribute that is an identifier (assigned by the user) of the application. The optional "project" attribute defines a name of a project within which a job was submitted. The "commitWait" attribute specifies whether a job must be committed before execution. If commitWait is set to 'true' a submitted job will not be executed until it is committed. Otherwise it is executed without waiting for a commit. The default value is 'false'. The <grmsjob> element contains an optional element <jobNote> and a mandatory list of <task> elements.

<jobNote> contains an arbitrary job description. It can be used by users or client software to store specific information about a job.

<task> this element is used for describing a single task, which is generally a definition of executable together with a set of parameters (executable parameters, standard input, output and error streams, environment variables etc.) and resource requirements needed for its appropriate execution. It contains one mandatory attribute and four optional ones. The mandatory "taskid" attribute specifies an identifier of a task. This identifier must be unique within the job description document. If the "persistent" attribute is set on 'true' GRMS does not remove task's working directory after its completion. The default value is 'false'. In the "extension" attribute one can specify such a task that this task will be executed in its working directory. The "crucial" attribute determines whether a failure of this task should cause a failure of the whole job(workflow). The default value is 'true'. The "commitWait attribute has the same meaning as in the <grmsJob> element. If commitWait is set to true a submitted task will not be executed until it is committed. The default value is 'false'. The <task> element has to contain the mandatory <executable> element and optionally the following elements: <taskNote>, <resource>, <hardConstraints>, <softConstraints>, <executionTime>, and <workflow>.

<taskNote> contains an arbitrary task description. It can be used by users or client software to store specific information about a task.

<resource> this element is used to describe resource requirements for execution of a single task. There might be more than one <resource> element. The 'OR' logical operator is used for these elements during a resource discovery process. It means that resource requirements are satisfied if a resource description matches requirements specified in at least one of <resource> elements. Resource description can contain the following information:

<ostype> type of the operating system

<osname> name of the operating system,

<osversion> version of the operating system,

<osrelease> release of the operating system,

<hostname> name of the host where job should be executed. It may contain one optional "tileSize" attribute that defines how much processes of parallel application must be executed at this host. Defining multiple <resource> elements a user can define explicitly allocation of a parallel application among hosts.

<localrmname> local resource management system available at the host. Acceptable values are: "fork" (default value), "lsf", "pbs", "sge", “condor”, "ccs", and "queue" (arbitrary queueing system),

<memory> minimal amount of the memory in MB,

<cpucount> minimal number of processors,

<cpuspeed> minimal speed of CPU(s) (in MHz),

<bandwidth> is a measure for the amount of network bandwidth that is "unused" or in other words available between two hosts (in MBs). It contains the "hostname" attribute which defines a second host,

<latency> denotes the minimal time required to send a message to another host (in seconds). It contains the "hostname" attribute which defines a second host,

<capacity> denotes capacity network between two locations determines the maximum throughput that you can achieve (the capacity of the entire route is determined by the link with the lowest capacity) (in MBs). It contains the "hostname" attribute which defines a second host,

<applications> is a list of required applications that have to be installed on a destination host. Every single <application> element contains two optional attributes: "version" and "instanceCount".

The "version" attribute denotes a required version of an application.

The "instanceCount" attribute defines a required number of application instances that must be started in order to execute the main job. A default value is 1.

<freememory> defines a minimal amount of free memory in MB,

<diskspace> defines a minimal amount of disk space in MB,

<freediskspace> defines a minimal amount of free disk space in MB,

<queue> specifies a name of a queue to which a job has to be submitted,

<freecpus> defines a minimal number of free CPUs.

Important

Please note, that if the application is not "installed" on all remote hosts, resource requirements have to contain information at least about an operating system, which the executable was compiled for, or a name of machine where the application should be executed.

<hardConstraints> allows defining advanced constraints. It can be used if user's requirements are too complex to be expressed in the <resource> element. It contains a list of <constraint> elements.

<constraint> defines a single constraint concerning computational resources (nodes, clusters etc.) that must be met (e.g. number of CPUs in range <4,32>). The optional "indiffThreshold" attribute defines the maximal difference between a required value and an actual value of a parameter such that a given resource satisfies a constraint (e.g. for the constraint: Memory > 50MB and indiffThreshold=5MB, a resource providing Memory=45MB satisfies this constraint while a resource providing Memory=44MB does not)

<parameter> defines a constraint imposed on a certain parameter. The required "name" attribute specifies a parameter on which the constraint is imposed (e.g. CPUspeed). The name must define a parameter supported by GRMS. It is not case sensitive. It contains lists of <value> and <range> elements.

The 'OR' logical operator is used to evaluate the whole constraint. It means that a constraint is satisfied if the required value is equal at least to one of specified values or belongs to at least one of specified ranges.

<value> defines the exact required value. A constraint is satisfied if it is equal to a value of a parameter provided by a particular resource (taking into account the indifference threshold)

<range> defines the minimum and maximum values using the optional "min" and "max" attributes. If min or max value is not defined -infinity and infinity are taken as default respectively. Thus if neither min nor max values are defined all real values satisfy this constraint.

<endpoint> defines an endpoint for network parameters. For instance, it may be used for a definition of a minimum bandwidth between a destination host and a host specified in this element.

<softConstraints> expresses user's preferences needed by a resource broker to select the best resources for a task (for example, to find a machine with the lowest CPU load and the greatest amount of free memory assuming that CPU load is two times more important than free memory). It has one mandatory attribute: "preferenceType".

The "preferenceType" attribute determines a method of expressing user's preferences. Two methods are currently supported:

- `PRIORITY` - value of the <importance> element denotes a numeric measure of the constraint's importance (e.g. if this value is two times greater than for another constraint then this constraint is two times more important)
- `RANKING` - value of the <importance> element denotes a position of this soft constraint in the ranking of all soft constraints (e.g. CPU load is the second most important constraint)

It consists of a list of <constraint> elements.

<constraint> defines a single soft constraint and its importance. It contains three attributes. The mandatory "importance" attribute defines an importance of this soft constraint (according to the preferenceType defined in the <softConstraints> element). The optional "indiffThreshold" attribute has the similar meaning as for the <hardConstraints> element. It defines when values of parameters are considered equal.

The "optimizationType" attribute allows a user to decide whether a given parameter is to be minimized or maximized. The following values are supported:

- `GAIN` - the higher value of parameter means the better resource
- `COST`- the less value of parameter means the better resource

The <constraint> element consists of two sub-elements: the mandatory <parameter> and optional <endpoint> element.

<parameter> defines a name of a parameter on which the constraint is imposed (e.g. CPUspeed). The name must define a parameter supported by GRMS. It is not case sensitive.

<endpoint> defines an endpoint for network parameters. For instance, it may be used for a definition of a minimum bandwidth between a destination host and a host specified in this element.

<executable> describes the executable. It contains “type”, “count” and "checkpointable" attributes and must contain either <application> or <execfile> element. Optionally, it can contain the following single elements: <arguments>, <stdin>, <stdout>, <stderr>, <environment>, and <checkpoint>.

The “type” attribute specifies an executable type - a way in which the job-manager submits the job. The following values are available:

- single - even if the count > 1, only 1 process or thread will be started,
- multiple - runs a number of processes or threads defined by the "count" attribute,
- mpi - uses an appropriate method to run a task compiled with a vendor-provided MPI library. A task is started at a number of nodes defined by the "count" attribute.
- mpichg - runs a task across multiple sites using MPICH-G library. A task is started at a number of nodes defined by the "count" attribute.

The “count” attribute denotes a number of processes of the executable. A default value is 1.

The "checkpointable" attribute determines whether a job can be checkpointed or not. A default value is 'false'.

<application> defines an application that is to be executed. This application must be available (installed earlier) on a destination host. It may contain the "version" attribute.

<execfile> this element describes a file that is to be used as an executable. To this end, it contains, the "name" attribute, four additional attributes describing an operating system. The "name" attribute denotes a local name of a file after staging in. The remaining optional attributes "osname", "ostype", "osversion", "osrelease" defines an operating system that a given executable is designed for. It has to contain either <logicalId> or <url> element and may contain an optional "reference" element.

<logicalId> specifies a logical file identifier. GRMS uses this identifier to get a file from data management system. A user do not need to know URL location of a file. The system GRMS uses depends on GRMS configuration. It contains an optional "user" attribute that specifies an owner of a file in a data management system,

<url> denotes URL location of a file (gass http, grid ftp),

<reference> specifies a logical file reference to a file that is output of other task of this job. In this way a user do not need to specify exact file locations and names to use output of one task as an executable in another.

<arguments> defines arguments for a task execution. Each argument must be either <file>, <value>, or

<directory> element.

<file> this element describes a file that is input or output argument for this executable. It contains, two mandatory attributes: "name" and "type", three additional ones: "required", "append", and "permissions". The logical path is the concatenation of the “name” attribute and the value of tag.

- "name" - denotes a local name of a file after staging in or before staging out,
- "type" - specifies a type of a file that can have one of the values: "in" (for input file), "out" (for output file),
- "required" - specifies whether this file is required to run this task. A default is 'true' (e.g. a task will not be executed if this file is not available),
- "append" - determines if existing file with the same name should be overwritten or appended. A default is 'false' (e.g. a file will be overwritten),
- "permissions" - defines unix-like permissions that will be given to this file during creation

<logicalId> specifies a logical file identifier. GRMS uses this identifier to get this file from or put it to a data management system. A user do not need to know URL location of a file. The system GRMS uses depends on GRMS configuration. It contains an optional "user" attribute that specifies an owner of a file in a data management system,

<url> denotes URL location of a file (gass http, grid ftp),

<reference> specifies a logical file reference to a file that is output or input of other task of this job. In this way a user do not need to specify exact file locations and names to use output of one task as an input in another.

<value> describes command line arguments of the executable.

<directory> can be used to describe a directory, which is needed for the execution of this task or has to be transferred after this task is done. It contains the same set of attributes and elements as the <file> element.

<stdin> denotes standard input for the executable. It contains the same elements as the <file> element,

<stdout> denotes standard output for the executable. It contains the same elements as the <file> element,

<stderr> denotes standard error for the executable. It contains the same elements as the <file> element,

<environment> can be used to describe environment variables for a task execution. Each <variable> element contains a variable's value while the attribute "name" denotes a name of this variable,

<checkpoint> has to be used for a job description for the migration call. It describes application's checkpoint files and directories. It has to contain one or more <file> or <directory> elements (for details see description of these elements for <execfile> element).

<executionTime> defines time constraints that are taken into account during job scheduling. It must contain the <execDuration> element and it may contain the <timeSlot> and <timePeriod> elements.

<timeSlot> defines a slot within a day when a job must be executed (e.g. between 10.00AM and 4.00PM). It must include the <slotStart> element and either the <slotEnd> or <slotDuration> elements.

- <slotStart> specifies start time of the slot (as time of a day). A task must be started after that time.
- <slotEnd> specifies end time of the slot (as time of a day). A job must be started before that time.
- <slotDuration> specifies duration time of a slot. A task must be started before slot duration time ends.

<execDuration> specifies execution time of a task in minutes (i.e. it defines length of the period when a resource reservation is needed for a task).

<timePeriod> defines a time period when a task must be executed (e.g. between Monday and Friday). It must contain either the <periodEnd> or <periodDuration> elements. It may also include the following optional elements: <periodStart>, <excluding>, and <including>.

- <periodStart> specifies start time of a period during which a task must be started (e.g. 31st January 10.00AM).
- <periodEnd> specifies the end of a period (e.g. 12th February 2005 16:00PM).
- <periodDuration> specifies duration of the time period (e.g. one week). If <periodStart> is not specified a default value of a period start time is a current time.
- <including> restricts a period when a task can be executed to certain days of a week (using the <weekDay> element) and/or dates (using the <dateDay> element), e.g. execute a job only on Fridays.
- <excluding> excludes certain days of a week (using the <weekDay> element) and/or dates (using the <dateDay> element) from a period when a task can be executed, e.g. do not execute jobs on Sunday.

<workflow> defines a workflow of tasks. It must contain a list of parents. It can also have the optional "parentStates" attribute which specifies whether this task has to be run after all parents meet required states ("AND" value) or any of parents meets a required state ("OR" value). "AND" is a default value.

- <parent> specifies a parent of this task. It can have two optional attributes: "triggerState" and "runSameHost". The former defines such a parent state that if obtained by the parent this task can be started. A default value is 'FINISHED' which means that a task can be executed when its parent finishes. The latter specifies whether this task has to be run at the same host as its parent. A default value is 'false'.

GRMS supports the following environment variables that can be used in GRMS Job Description and will be replaced by system with proper values:

HOME the user’s HOME directory on the remote host,

JOB_ID the job identifier,

TASK_ID the task identifier,

TASK_DIR the task's working directory

JAVA_HOME the location of the Java on the remote host.

HOSTNAME name of the host which the job is/was executed on.

JobDescription examples

Example 4.1. The simplest case

The simplest example of a job that can be executed by GRMS, is the job consisting of one task, which describes an application that does not need any arguments, has no resource requirements and its user is not interested in catching stdout and/or stderr. Let's assume it is the **/bin/date** program, which should be available on each unix/linux platform. Please take a note of the number of slashes in the file url. The amount of slashes is caused by inconsistency between RFC 1738, which defines the file:// url, and the GlobusURL class, which implements this norm. In RFC, the root directory is accessible by typing 3 slashes, but the GlobusURL needs four ones.

```
<grmsJob appid="example1">
   <task taskid="date">
      <executable type="single" count="1">
         <execfile name="exec-file">
            <url>file:////bin/date</url>
         </execfile>
      </executable>
   </task>
</grmsJob>
```

Example 4.2. Host name specification

If the **/bin/date** should be executed on a specific machine the GRMS Job Description has to be extended by adding <resource> and <hostname> tags.

```
<grmsJob appid="example2">
   <task taskid="date">
      <resource>
         <hostname>rage1.man.poznan.pl</hostname>
      </resource>
      <executable type="single" count="1">
         <execfile name="exec-file">
            <url>file:////bin/date</url>
         </execfile>
      </executable>
   </task>
</grmsJob>
```

In this case, **/bin/date** will be executed on the host `rage1.man.poznan.pl`.

Example 4.3. Resource description

If the **/bin/date** should be executed on any linux machine, which has at least 2 CPUs. The jobDescription should look as follow:

```
<grmsJob appid="example3">
   <task taskid="date">
      <resource>
         <osname>Linux</osname>
         <cpucount>2</cpucount>
      </resource>
      <executable type="single" count="1">
```

```
         <execfile name="exec-file">
            <url>file:////bin/date</url>
         </execfile>
      </executable>
   </task>
</grmsJob>
```

Example 4.4. Physical location

If the execution of task should be preceded by getting an executable file from a specific location (defined by the user), the <url> tag should be used appropriately. Let's assume that the file has the name "date" and it is placed in the "examples" directory, which is the subdirectory of the home directory of the user. The example <url> looks as follows; `gsiftp://rage1.man.poznan.pl/~/examples/date`. ("~" denotes your working directory). You can also specify the whole gsiftp path to your executable (`gsiftp://rage1.man.poznan.pl//home/user1/examples/date`). Note that there are two slashes after the address of machine.

```
<grmsJob appid="example4">
   <task taskid="date">
      <executable type="single" count="1">
         <execfile name="exec-file">
            <url>gsiftp://rage1.man.poznan.pl/~/examples/date</url>
         </execfile>
      </executable>
   </task>
</grmsJob>
```

Example 4.5. Logical location

If the execution of task should be preceded by getting an executable file from the Gridge Data Management System, the <logicalId> tag should be used to specify logical identifier of file.

```
<grmsJob appid="example5">
   <task taskid="data">
      <executable type="single" count="1">
         <execfile name="exec-file">
            <logicalId>382</logicalId>
         </execfile>
      </executable>
   </task>
</grmsJob>
```

Example 4.6. Arguments

If the executable needs some input arguments (typically passed as command line arguments) they can be passed as <value>s in the <arguments> section. GRMS Job Description for "**/bin/echo** *`Hello World`*" looks as follow:

```
<grmsJob appid="example6">
   <task taskid="echo">
      <executable type="single" count="1">
         <execfile name="exec-file">
            <url>file:////bin/echo</url>
         </execfile>
         <arguments>
            <value>Hello </value>
            <value>World</value>
         </arguments>
      </executable>
   </task>
</grmsJob>
```

Example 4.7. Files staging in

If a program needs for its proper execution some files to be copied into a working directory it can be done by using files of the “in” type. Let’s assume the user wants to execute “**/bin/cat** *file.log*”, where file.log is the file which should be copied first. It can be specified in the following way:

```
<grmsJob appid="example7">
   <task taskid="cat">
      <executable type="single" count="1">
         <execfile name="exec-file">
            <url>file:////bin/cat</url>
         </execfile>
         <arguments>
            <value>file.log</value>
            <file name="file.log" type="in">
               <url>gsiftp://rage1.man.poznan.pl/~/examples/file.log</url>
            </file>
         </arguments>
      </executable>
   </task>
</grmsJob>
```

Example 4.8. Files staging out

If a program generates, as a result of its execution, files that have to be transferred to some locations they can be defined in the <arguments> section as files of the “out” type. Let’s assume that we want to compress the file “report” using tar and then the created archive should be copied to the location: “`gsiftp://fury.man.poznan.pl/~/examples/report.tar`”.

```
<grmsJob appid="example8">
   <task taskid="tar">
      <executable type="single" count="1">
         <execfile name="exec-file" type="in">
            <url>file:////bin/tar</url>
         </execfile>
         <arguments>
            <value>cfv</value>
            <value>file.tar</value>
            <value>report</value>
            <file name="report" type="in">
               <url>gsiftp://rage1.man.poznan.pl/~/examples/report</url>
            </file>
            <file name="report.tar" type="out">
               <url>gsiftp://rage1.man.poznan.pl/~/examples/report.tar</url>
            </file>
         </arguments>
      </executable>
   </task>
</grmsJob>
```

Example 4.9. Directories staging in

If an application for its execution needs a directory to be staged in the location (logical or physical) can be expressed using <directory> section.

```
<grmsJob appid="example9">
   <task taskid="dir" persistent="true">
      <executable type="single" count="1">
         <execfile name="exec-file">
            <url>file:////bin/ls</url>
         </execfile>
         <arguments>
            <value>-la</value>
            <value>examples</value>
            <directory name="examples" type="in">
               <url>gsiftp://fury.man.poznan.pl/~/examples</url>
```

```
            </directory>
         </arguments>
         <stdout>
            <url>${TASK_DIR}/DIR.txt</url>
         </stdout>
      </executable>
   </task>
</grmsJob>
```

Example 4.10. Standard input

If a program needs to read some data from a file to stdin it can be specified by the <stdin> tag. Let's assume that we want to execute the **/bin/cat** reading stdin from the logical file with id 383, the output should be stored in "`stdout_file`". In this case contents of logical file should be copied to "`stdout_file`".

```
<grmsJob appid="example10">
   <task taskid="cat">
      <executable type="single" count="1">
         <execfile name="exec-file">
            <url>file:////bin/cat</url>
         </execfile>
         <stdin>
            <logicalId>383</logicalId>
         </stdin>
         <stdout>
            <url>gsiftp://fury.man.poznan.pl/~/examples/stdout_file</url>
         </stdout>
      </executable>
   </task>
</grmsJob>
```

Example 4.11. Standard output

If a standard output (stdout) of executed application should be stored in a location defined by the user, it can be done by using the <stdout> tag. Let's assume that we want to execute the application (e.g. grep), which finds in a given input file all lines containing the string "GRMS" and puts them to the logical file.

```
<grmsJob appid="example11">
   <task taskid="grep">
      <executable type="single" count="1">
         <execfile name="exec-file">
            <url>file:////bin/grep</url>
         </execfile>
         <arguments>
            <value>GRMS</value>
            <value>grep_input</value>
            <file name="grep_input" type="in">
               <url>gsiftp://fury.man.poznan.pl/~/examples/grep_input</url>
            </file>
         </arguments>
         <stdout>
            <logicalId>384</logicalId>
         </stdout>
      </executable>
   </task>
</grmsJob>
```

Example 4.12. Environment variables

If a program requires some environment variables to be setup first, it can be done by using <environment> and <variable> tags. Let's assume that we want to set GRMS environment variable to the "GRMS Example" and then display it using a script containing the "**/bin/echo** `$GRMS`". Additionally the <stdout> should be redirected to the specified file. In this case, an example of GRMS Job Description might look as follow:

```
<grmsJob appid="example12">
   <task taskid="echo">
      <executable type="single" count="1">
         <execfile name="exec-file">
            <url>gsiftp://fury.man.poznan.pl/~/examples/echo.sh</url>
         </execfile>
         <stdout>
            <url>gsiftp://fury.man.poznan.pl/~/examples/echo_out</url>
         </stdout>
         <environment>
            <variable name="GRMS">GRMS Example</variable>
         </environment>
      </executable>
   </task>
</grmsJob>
```

Example 4.13. GRMS variables

This example presents the usage o GRMS environment variables. Output of the date command will be stored in file that name contains job and task identifiers.

```
<grmsJob appid="example13">
   <task taskid="date">
      <resource>
         <hostname>fury.man.poznan.pl</hostname>
      </resource>
      <executable type="single" count="1">
         <execfile name="date">
            <url>file:///bin/date</url>
         </execfile>
         <stdout>
            <url>gsiftp://fury.man.poznan.pl/~/examples/date-${JOB_ID}-${TASK_ID}</url>
         </stdout>
      </executable>
   </task>
</grmsJob>
```

Example 4.14. Suspend/Resume

This example shows details connected with suspend/resume and migrate task functionalities. Task has to be marked as checkpointable and checkpoint section has to contain information needed to stage out files during task suspending and to stage them in before resuming the task..

```
<grmsJob appid="example14">
   <task taskid="suspend">
      <resource>
         <hostname>fury.man.poznan.pl</hostname>
      </resource>
      <executable type="single" count="1" checkpointable="true">
         <execfile name="chkpt_test">
            <url>gsiftp://fury.man.poznan.pl/~/examples/chkpt_test</url>
         </execfile>
         <arguments>
            <value>6000</value>
         </arguments>
         <checkpoint>
            <file name="checkpoint" type="out">
               <url>gsiftp://fury.man.poznan.pl/~/examples/checkpoint-${JOB_ID}</url>
            </file>
            <file name="checkpoint" type="in">
               <url>gsiftp://fury.man.poznan.pl/~/examples/checkpoint-${JOB_ID}</url>
            </file>
         </checkpoint>
      </executable>
   </task>
</grmsJob>
```

Example 4.15. Persistent/Extension

This example presents the usage of "persistent" and "extension" attributes. Let assume that we want to pack (tar) the output of "**/bin/ps** *-ef*" command and copy the obtained archive to specified location. To avoid transferring data the second task should be executed in the same directory as the first one was. The whole experiment will be divided into two parts: first task will execute aforementioned command and catch its output to specified file (the task will be specified as a persistent one, so grms will not remove its directory when it finished), then next task will pack the file and copy to remote location (the second task will be specified as a extension of the fist one, so will be executed in the same directory as the first one). The second task can start only when the first one will be finished.

```
<grmsJob appid="example15">
   <task taskid="ps" persistent="true">
      <resource>
         <hostname>fury.man.poznan.pl</hostname>
      </resource>
      <executable type="single" count="1">
         <execfile name="ps">
            <url>file:///bin/ps</url>
         </execfile>
         <arguments>
            <value>-ef</value>
         </arguments>
         <stdout>
            <url>${TASK_DIR}/ps.out</url>
         </stdout>
      </executable>
   </task>
   <task taskid="tar" extension="ps">
      <executable type="single" count="1">
         <execfile name="tar">
            <url>file:///bin/tar</url>
         </execfile>
         <arguments>
            <value>cfz</value>
            <value>ps.out.tgz</value>
            <value>ps.out</value>
            <file name="ps.out.tgz" type="out">
               <url>gsiftp://fury.man.poznan.pl/~/examples/ps.out.tgz</url>
            </file>
         </arguments>
      </executable>
      <workflow>
         <parent triggerState="FINISHED">ps</parent>
      </workflow>
   </task>
</grmsJob>
```

Example 4.16. Data references

Let's imagine a more complicated experiment. Let assume that we want to execute **"/bin/ps** *-ef"* command on `rage1` and `rage2` machines (tasks: rage1_ps and rage2_ps) and then pack caught outputs on `rage3`, where the tar application is installed. Final archive file should be copied to the specified location on `rage4` machine. The simplest way to express this experiment is to use data references functionality that give the user possibility to express that output data generated by one task can be the input data for the other one. In this case execution of rage1_ps and rage2_ps tasks can be done simultaneously and the task rage3_tar can be executed only when "ps" tasks will finish.

```
<grmsJob appid="example16">
   <task taskid="rage1_ps">
      <resource>
         <hostname>rage1.man.poznan.pl</hostname>
      </resource>
      <executable type="single" count="1">
         <execfile name="ps">
            <url>file:///bin/ps</url>
         </execfile>
         <arguments>
            <value>-ef</value>
```

```
         </arguments>
         <stdout>
            <reference>rage1_ps_output</reference>
         </stdout>
      </executable>
   </task>
   <task taskid="rage2_ps">
      <resource>
         <hostname>rage2.man.poznan.pl</hostname>
      </resource>
      <executable type="single" count="1">
         <execfile name="ps">
            <url>file:////bin/ps</url>
         </execfile>
         <arguments>
            <value>-ef</value>
         </arguments>
         <stdout>
            <reference>rage2_ps_output</reference>
         </stdout>
      </executable>
   </task>
   <task taskid="fury_tar" persistent="true">
      <resource>
         <hostname>fury.man.poznan.pl</hostname>
      </resource>
      <executable type="single" count="1">
         <execfile name="tar">
            <url>file:////bin/tar</url>
         </execfile>
         <arguments>
            <file name="rage1_ps.out" type="in">
               <reference>rage1_ps_output</reference>
            </file>
            <file name="rage2_ps.out" type="in">
               <reference>rage2_ps_output</reference>
            </file>
            <value>cfz</value>
            <value>example15.tgz</value>
            <value>rage1_ps.out</value>
            <value>rage2_ps.out</value>
            <file name="example15.tgz" type="out">
               <url>gsiftp://fury.man.poznan.pl/~/examples/example15-${JOB_ID}.tgz</url>
            </file>
         </arguments>
         <stdout>
            <url>gsiftp://fury.man.poznan.pl/~/examples/stdout-${JOB_ID}</url>
         </stdout>
         <stderr>
            <url>gsiftp://fury.man.poznan.pl/~/examples/stderr-${JOB_ID}</url>
         </stderr>
      </executable>
      <workflow parentStates="AND">
         <parent triggerState="FINISHED">rage1_ps</parent>
         <parent triggerState="FINISHED">rage2_ps</parent>
      </workflow>
   </task>
</grmsJob>
```

Example 4.17. Notes

The whole job as well as each task can have human readable descriptions inside <JobNote> and <taskNode> sections.

```
<grmsJob appid="example17">
   <jobNote>
     job description
   </jobNote>
   <task taskid="date">
      <taskNote>
        task description
      </taskNote>
      <executable type="single" count="1">
         <execfile name="exec-file">
            <url>file:////bin/date</url>
         </execfile>
      </executable>
```

```
   </task>
</grmsJob>
```

Example 4.18. Files/Directories properties

For every file to be staged in or out it is possible to specify if it is required, if it should be appended to the existing file or should overwrite it and file permissions.

```
<grmsJob appid="example18">
   <task taskid="date">
      <executable type="single" count="1">
         <execfile name="exec-file">
            <url>file:////bin/date</url>
         </execfile>
         <arguments>
            <file name="date.out" type="out" required="false" append="true" permissions="666">
               <url>gsiftp://fury.man.poznan.pl/~/examples/date.append</url>
            </file>
         </arguments>
         <stdout>
            <url>${TASK_DIR}/date.out</url>
         </stdout>
      </executable>
   </task>
</grmsJob>
```

Example 4.19. Crucial tasks

Using the "crucial" attribute it is possible to mark some tasks as "unimportant" from the job's point of view (setting this property to "false"). Failure of such task doesn't cause the failure of the whole job.

```
<grmsJob appid="example19">
   <task taskid="ps">
      <resource>
         <hostname>fury.man.poznan.pl</hostname>
      </resource>
      <executable type="single" count="1">
         <execfile name="ps">
            <url>file:////bin/ps</url>
         </execfile>
         <arguments>
            <value>-ef</value>
         </arguments>
         <stdout>
            <url>${TASK_DIR}/ps.out</url>
         </stdout>
      </executable>
   </task>
   <task taskid="date" crucial="false">
      <executable type="single" count="1">
         <execfile name="date">
            <url>file:////bin/tar</url>
         </execfile>
         <arguments>
            <file name="unnecessary_input" type="in">
               <url>gsiftp://fury.man.poznan.pl/~/examples/UNEXISTING_FILE</url>
            </file>
         </arguments>
      </executable>
   </task>
</grmsJob>
```

Example 4.20. Time constraints

If a user is available only in a certain time period he/she can express his/her time constraints using the

<executionTime> element. In this example a task will be executed after 11th August 2006 and before 15th August 2006, with exception of Sundays, between 10am and 4pm.

```
<executionTime>
    <timeSlot>
        <slotStart>10:00:00</slotStart>
        <slotEnd>16:00:00</slotEnd>
    </timeSlot>
    <execDuration>PT30M</execDuration>
    <timePeriod>
        <periodStart>2006-08-11T00:00:00.000+02:00</periodStart>
        <periodEnd>2006-08-15T00:00:00.000+02:00</periodEnd>
        <excluding>
             <weekDay>Sunday</weekDay>
        </excluding>
    </timePeriod>
</executionTime>
```

Example 4.21. Advanced constraints

In addition to simple resource requirements defined in the <resource> section, a user can also express more complex requirements (hard constraints) and preferences concerning a selection of the best resources (soft constraints). In this example a user requires a resource that provides either 4 or 8 or any number within the range [16;32] processors. Another requirement is at least 100MB of free memory. Moreover a user asks a resource broker to select a resource so that amount of free memory is maximized and CPU load minimized. Large amount of free memory is two times more important that CPU load for this task.

```
<hardConstraints>
   <constraint>
      <parameter name="cpuCount">
         <value>4</value>
         <value>8</value>
         <range min="16" max="32"/>
      </parameter>
   </constraint>
   <constraint>
      <parameter name="freeMemory">
         <range min="100"/>
      </parameter>
   </constraint>
</hardConstraints>
<softConstraints preferenceType="PRIORITY">
   <constraint importance="2" optimizationType="GAIN">
      <parameter>freeMemory</parameter>
   </constraint>
   <constraint importance="1" optimizationType="COST">
      <parameter>cpuLoad</parameter>
   </constraint>
</softConstraints>
```

Chapter 5. GRMS java client

Requirements

- Installation
  - sed
  - perl (optionally Term::ReadLine perl module)
- Usage
  - Java - JRE 1.4.x

Installation and configuration of GRMS client

Grms client doesn't need any extra privileges and can be installed for all users by administrator or every user can have his own instance.

To install and configure the grms client please follow the procedure described bellow:

1. download "grms-install-client.pl" from `www.gridge.org/grms`
2. start installation replacing grms-client-path with real path you want grms client to be installed

   $> **grms-install-client.pl** *--dest <grms-client-path>*
3. Answer all questions needed to configure the client

```
piontek@druid-bis ~/GRMS-client $ ./grms-install-client.pl --dest .
directory '/home/piontek/GRMS-client/.' already exists, do you want to continue (y/n)? y
do you want to remove directory '/home/piontek/GRMS-client/.' first (y/n) ? n
extracting distribution ...
setup files ...
Enter default GRMS server URL [https://druid-bis.man.poznan.pl:8753/axis/services/grms]: \
                                    https://druid-bis.man.poznan.pl:8342/axis/services/grms
Enter default GRMS server distinguished name []: \
                                          /C=PL/O=GRID/O=PSNC/CN=grms/druid-bis.man.poznan.pl
Enter default GRMS server delegation type [FULL|LIMITED|NO] [FULL]: FULL
Enter default GRMS server timeout (min) [5]:
installation successfully completed
```

4. Global configuration of grms client can be changed by modifying properties set in `<grms-client-path>/bin/grms-client` script

   Please modify following lines if necessary:

```
GRMS_URL=”https://druid-bis.man.poznan.pl:8342/axis/services/grms”
GRMS_DN="/C=PL/O=GRID/O=PSNC/CN=grms/druid-bis.man.poznan.pl"
GRMS_DELEG_TYPE=FULL
GRMS_TIMEOUT=5
```

   - `GRMS_URL` describes location of GRMS service.
   - `GRMS_DN` describes the expected by the client distinguish name of the service. If GRMS_DN is empty the client doesn't authorize the service and is not able to delegate user proxy to the service,
   - `GRMS_TIMEOUT` property sets the timeout (in minutes) for service response.

- `GRMS_DELEG_TYPE` – determines the type of proxy delegation. Following values are allowed: FULL, LIMITED, NO.

5. Every user before launching grms client has to configure CoG library creating or modifying `cog.properties` file in `~/.globus` directory. Please change values of properties according to your configuration

```
#Java CoG Kit Configuration File

usercert=/home/piontek/.globus/usercert.pem
userkey=/home/piontek/.globus/userkey.pem
proxy=/tmp/x509up_u501
cacert=/etc/grid-security/certificates/
```

   For details please visit: CoG Home Page [???]

6. If you doesn't want to install your own instance of client but you prefer to use a common one with your own configuration you can do this using profiles. Every GRMS user can do define set of profiles (different client configurations) storing them in ~/.grms_profiles file. Every profile has to be in separate line with parameters separated by tabulators. Every line has to follow the format:

```
profile_name    GRMS_URL    GRMS_DN    GRMS_DELEG_TYPE    GRMS_TIMEOUT
```

   e.g.

```
druid   https://druid-bis.man.poznan.pl:8343/axis/services/grms \
        /C=PL/O=GRID/O=PSNC/CN=grms/druid-bis.man.poznan.pl  FULL  5
```

GRMS command line client

Executing GRMS Client

1. source the <*grms-client-path*>`/grms-env.sh` file to setup needed environment variables,

2. make sure that the user's proxy exists. If it doesn't please create or download it using one of following methods:

   - run **`$GLOBUS_LOCATION`/bin/grid-proxy-init** command to create user's proxy

   - run **`$GLOBUS_LOCATION`/bin/myproxy-get-delegation** command to download user's proxy from myproxy-server

3. run the client
   `grms-client` [-s *`profile_name`*] *`operation`* [*`argument`*...]

   If the optional profile is not specified the global client configuration will be used.

   e.g.

```
./grms-client -s myprofile submit_job ../examples/example1.xml

./grms-client submit_job ../examples/example1.xml
```

Operations

The GRMS client supports following list of operations:

- *submit_job* - submits new job to the system,
- *commit_job* - commits job submitted with two phase commit property,
- *recover_job* - recovers job after failure, skips previously completed tasks,
- *suspend_job* - suspends job suspending all running tasks,
- *resume_job* - resumes previously suspended job,
- *cancel_job* - cancels whole job (running tasks are killed),
- *refresh_job* - refreshes user's proxy associated with job,
- *migrate_task* - migrates simple task to better resource,
- *suspend_task* - suspends single task,
- *resume_task* - resumes single task,
- cancel_task - cancels single task,
- *list_all* - lists all jobs in the system in specified state,
- *list_jobs* - list all jobs or jobs in specified state belonging to the user,
- *list_tasks* - lists all tasks or tasks in specified state,
- *list_project* - lists all jobs or jobs belonging to given project,
- *register_access* - registers location of web service interface that can be used to checkpoint specific task,
- *unregister_access* - unregisters interface for checkpoining,
- *get_access* - returns location of registered interface for checkpointing,
- *job_info* - returns complex information about given job,
- *task_info* - returns complex information about given task,
- *resources* - lists resources that meet user's requirements,
- *test* - tests syntax and logical correctness of job description,
- *add_job_notif* - registers notification concerning changes of status of given job,
- *del_job_notif* - unregisters notification concerning changes of status of given job,
- *get_job_notif* - lists notification registered for given job,
- *add_task_notif* - registers notification concerning changes of status of given task or request processing,
- *del_task_notif* - unregisters task notification,
- *get_task_notif* - lists informations concerning registered notifications or returns description of given notification,
- *add_file_dir* - for a given task allows to register dynamically additional file or directory,
- *del_file_dir* - for a given task allows to unregister file or directory, concerns as well dynamically added files and directories as ones specified in job description,

- *get_file_dir* - for a given task returns list of registered files and directories that meet specified criteria,
- *extend_execution* - extends time of execution specified in job description,
- *description* - returns service description.

Usage syntax

```
JOB_STATUS = UNCOMMITED | SUBMITTED | ACTIVE | FINISHED | SUSPENDED | FAILED | CANCELED | BROKEN

TASK_STATUS = UNSUBMITTED | UNCOMMITTED | QUEUED | PREPROCESSING | PENDING | RUNNING | STOPPED |
              POSTPROCESSING | FINISHED | SUSPENDED | FAILED | CANCELED

TASK_REQUEST_STATUS = TASK_UNSUBMITTED | TASK_UNCOMMITED | TASK_QUEUED | TASK_RESOURCE |
                      TASK_RESOURCE_DONE | TASK_STAGE_IN | TASK_STAGE_IN_DONE | TASK_SUBMIT |
                      TASK_SUBMIT_DONE | TASK_EXEC | TASK_EXEC_PENDING | TASK_EXEC_ACTIVE |
                      TASK_EXEC_DONE | TASK_STAGE_OUT | TASK_STAGE_OUT_DONE | TASK_DONE |
                      TASK_FAILED | TASK_CANCEL | TASK_CANCEL_DONE | TASK_CANCEL_FAILED |
                      MIGRATE_QUEUED | MIGRATE_EXEC_SUSPEND | MIGRATE_EXEC_SUSPEND_DONE |
                      MIGRATE_RESOURCE | MIGRATE_RESOURCE_DONE | MIGRATE_STAGE_IN |
                      MIGRATE_STAGE_IN_DONE | MIGRATE_SUBMIT | MIGRATE_DONE | MIGRATE_FAILED |
                      MIGRATE_STAGE_OUT | MIGRATE_STAGE_OUT_DONE | REQUEST_FAILED

FILE_DIR_LOGICAL_TYPE = ARGUMENT_IN | ARGUMENT_OUT | CHECKPOINT_IN | CHECKPOINT_OUT

FILE_DIR_ORIGIN_TYPE = DESCRIPTION | ADDED


grms-client submit_job <jobDescriptionFile>
grms-client commit_job <jobID>
grms-client recover_job <jobID> [<jobDescriptionFile>]
grms-client suspend_job <jobId>
grms-client resume_job <jobId> [<jobDescriptionFile>]
grms-client cancel_job <jobId>
grms-client refresh_job <jobId>

grms-client migrate_task <jobId> <taskId> [<jobDescriptionFile>]
grms-client suspend_task <jobId> <taskId>
grms-client resume_task <jobId> <taskId> [<jobDescriptionFile>]
grms-client cancel_task <jobId> <taskId>
grms-client commit_task <jobId> <taskId>

grms-client list_all <JOB_STATUS>

grms-client list_jobs [<JOB_STATUS>] [<limit>]

grms-client list_tasks <jobId> [<TASK_STATUS>]

grms-client list_project <project> [<JOB_STATUS>] [<limit>]


grms-client register_access <jobId> <taskId> <service_location> <pid>
grms-client unregister_access <jobid> <taskId>
grms-client get_access <jobId> <taskId>


grms-client job_info <jobId>
grms-client task_info <jobId> <taskId> [<history_limit>]


grms-client resources <taskId> <jobDescriptionFile>


grms-client test <jobDescriptionFile>


grms-client add_job_notif <jobid> SOAP <destination> [ <JOB_STATUS>[,<JOB_STATUS>]* ]
grms-client add_job_notif <jobid> GASS <destination> TRUE|FALSE \
                          [ <format> [ <JOB_STATUS>[,<JOB_STATUS>]* ]]
grms-client del_job_notif <jobid> <notificationId>
grms-client get_job_notif <jobid>
grms-client get_job_notif <jobid> <notificationId>


grms-client add_task_notif <jobid> <taskId> STATUS SOAP <destination> \
                                  [ <TASK_STATUS>[,<TASK_STATUS>]*]]

grms-client add_task_notif <jobid> <taskId> STATUS GASS <destination> TRUE|FALSE \
                                  [<format> [ <TASK_STATUS>[,<TASK_STATUS>]*]]
```

```
grms-client add_task_notif <jobid> <taskId> REQUEST SOAP <destination> \
                                  [ <TASK_REQUEST_STATUS>[,<TASK_REQUEST_STATUS>]*]

grms-client add_task_notif <jobid> <taskId> REQUEST GASS <destination> TRUE|FALSE \
                                  [<format> [ <TASK_REQUEST_STATUS>[,<TASK_REQUEST_STATUS>]*]]


grms-client add_tasks_notif <jobid> STATUS SOAP <destination> [ <TASK_STATUS>[,<TASK_STATUS>]*]]

grms-client add_tasks_notif <jobid> STATUS GASS <destination> TRUE|FALSE \
                                   [<format> [ <TASK_STATUS>[,<TASK_STATUS>]*]]

grms-client add_tasks_notif <jobid> REQUEST SOAP <destination> \
                                   [ <TASK_REQUEST_STATUS>[,<TASK_REQUEST_STATUS>]*]

grms-client add_tasks_notif <jobid> REQUEST GASS <destination> TRUE|FALSE
                                  [<format> [ <TASK_REQUEST_STATUS>[,<TASK_REQUEST_STATUS>]*]]

grms-client del_task_notif <jobid> <taskId> <notificationId>
grms-client get_task_notif <jobid> <taskId> [STATUS|REQUEST]
grms-client get_task_notif <jobid> <taskId> <notificationId> [STATUS|REQUEST]


grms-client add_file_dir <jobId> <taskId> <overwrite> FILE_DIR_LOGICAL_FILE <name> \
                                       PHYSICAL|LOGICAL <path> <user> FILE|DIRECTORY \
                                       <append> <required> [<permissions>]

grms-client get_file_dirs <jobId> <taskId> [FILE_DIR_LOGICAL_TYPE [,FILE_DIR_LOGICAL_TYPE]*
                                       FILE_DIR_ORIGIN_TYPE[,FILE_DIR_ORIGIN_TYPE]*]

grms-client del_file_dir <jobId> <taskId> ARGUMENT_IN|ARGUMENT_OUT|CHECKPOINT_IN|CHECKPOINT_OUT \
                                       <name> PHYSICAL|LOGICAL <path> <user> FILE|DIRECTORY

grms-client del_file_dirs <jobId> <taskId> FILE_DIR_LOGICAL_TYPE [,FILE_DIR_LOGICAL_TYPE]* \
                                       FILE_DIR_ORIGIN_TYPE[,FILE_DIR_ORIGIN_TYPE]*


grms-client extend_execution <jobId> <taskId> \
                            +|- <years> <months> <days> <hours> <minutes> <seconds>


grms-client description SHORT|FULL
```

Important

All parameters in square brackets are optional and can be omitted.

Examples of execution

General issues

Important

To simplify the syntax of some commands following definitions are used:

```
JOB_STATUS = UNCOMMITED | SUBMITTED | ACTIVE | FINISHED | SUSPENDED | FAILED |
             CANCELED | BROKEN

TASK_STATUS = UNSUBMITTED | UNCOMMITTED | QUEUED | PREPROCESSING | PENDING |
              RUNNING | STOPPED |POSTPROCESSING | FINISHED | SUSPENDED |
              FAILED | CANCELED

TASK_REQUEST_STATUS = TASK_UNSUBMITTED | TASK_UNCOMMITED | TASK_QUEUED |
             TASK_RESOURCE | TASK_RESOURCE_DONE | TASK_STAGE_IN |
             TASK_STAGE_IN_DONE | TASK_SUBMIT | TASK_SUBMIT_DONE | TASK_EXEC |
             TASK_EXEC_PENDING | TASK_EXEC_ACTIVE | TASK_EXEC_DONE |
             TASK_STAGE_OUT | TASK_STAGE_OUT_DONE | TASK_DONE | TASK_FAILED |
             TASK_CANCEL | TASK_CANCEL_DONE | TASK_CANCEL_FAILED |
             MIGRATE_QUEUED | MIGRATE_EXEC_SUSPEND | MIGRATE_EXEC_SUSPEND_DONE |
             MIGRATE_RESOURCE | MIGRATE_RESOURCE_DONE | MIGRATE_STAGE_IN |
             MIGRATE_STAGE_IN_DONE | MIGRATE_SUBMIT | MIGRATE_DONE |
             MIGRATE_FAILED | MIGRATE_STAGE_OUT | MIGRATE_STAGE_OUT_DONE |
             REQUEST_FAILED

FILE_DIR_LOGICAL_TYPE = ARGUMENT_IN | ARGUMENT_OUT | CHECKPOINT_IN | CHECKPOINT_OUT
```

```
FILE_DIR_ORIGIN_TYPE = DESCRIPTION | ADDED
```

- If the client is invoked with wrong parameters the information about the usage is displayed.
- If the user is not authorized to perform the requested operation the proper message is displayed and the request is not performed.

```
[piontek@druid-bis]$ grms-client suspend_job 1085556664951_appid_1620
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://druid-bis.man.poznan.pl:8442/axis/services/grms
- Job suspending failed!
- GRMS response was:
- errorCode: 501
- errorMessage: User is not authorized to perform this operation
```

- If the service is not able to perform the authorization the proper message is displayed and the service doesn't serve the request.

```
[piontek@druid-bis]$ grms-client cancel_job 1085556664951_appid_1620
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://druid-bis.man.poznan.pl:8442/axis/services/grms
- Cancelling of job failed!
- GRMS response was:
- errorCode: 500
- errorMessage: Authorization failed. Cannot contact Authorization Service: Connection refused
```

- If the user tries to perform the operation on non-existing job or task the proper error message is displayed.

```
[piontek@druid-bis]$ grms-client job_info non-existing-job-ID
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://druid-bis.man.poznan.pl:8442/axis/services/grms
- Getting job info failed!
- GRMS response was:
- errorCode: 114
- errorMessage: No such job in Job Repository
```

```
[piontek@druid-bis]$ grms-client task_info 1085556664951_appid_1620 non-existing-task-ID
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://druid-bis.man.poznan.pl:8442/axis/services/grms
- Getting job info failed!
- GRMS response was:
- errorCode: 117
- errorMessage: No such task in Task Repository
```

Important

If the GRMS service is configured to use GAS as an authorization decision point the service response in case of operation on unexisting job or task will be "User is not authorized to perform this operation".

- If the user tries to perform any operation on the job or task, which doesn’t belong to him, the proper error message is displayed.

```
[piontek@druid-bis]$ grms-client task_info 1083844628098_appid_2377
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://druid-bis.man.poznan.pl:8442/axis/services/grms
- Getting job info failed!
- GRMS response was:
- errorCode: 154
- errorMessage: User not authorized to perform operation - internal authorization
```

Important

For GAS authorization the response will be "User is not authorized to perform this operation".

- If the GRMS Job Description contains syntax error, the proper error message is displayed.

```
[piontek@druid-bis]$ grms-client submit_job ./examples/error.xml
```

```
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://druid-bis.man.poznan.pl:8442/axis/services/grms
- Job submission failed!
- GRMS response was:
- errorCode: 101
- errorMessage: Job description syntax error: The element type "value"
                must be terminated by the matching end-tag "</value>".
```

submit_job

`grms-client` submit_job *`jobDescription-file`*

Submits new job described by job description stored in file with given location. If the GRMS Job Description does not contain any syntax and logical errors, the identifier of the submitted job is displayed.

```
[piontek@druid-bis]$ grms-client submit_job ./examples/example2.xml
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://druid-bis.man.poznan.pl:8442/axis/services/grms
- Job submitted successfully, jobId=1085556664951_appid_1620
```

commit_job

`grms-client` commit_job *`jobId`*

Commits job submitted with two phase commit option (commitWait attribute)

```
[piontek@druid-bis]$ grms-client commit_job 1084456104272_appid_0728
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://druid-bis.man.poznan.pl:8442/axis/services/grms
- job committed successfully
```

If the job doesn't wait for commit, the proper error message is displayed.

```
[piontek@druid-bis]$ grms-client commit_job 1085141911671_appid_3916
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://druid-bis.man.poznan.pl:8442/axis/services/grms
- Job committing failed!
- GRMS response was:
- errorCode = 176
- errorMessage = Job already committed
```

recover_job

`grms-client` recover_job *`jobId`* [*`jobDescription-file`*]

Recovers job after the failure resubmiting it and skipping previously finished tasks. Optionally it is possible to specify new job description. If the description is valid new identifier of job is returned.

```
[piontek@druid-bis]$ grms-client recover_job 1085141911671_appid_3916 ./examples/example2b.xml
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://druid-bis.man.poznan.pl:8442/axis/services/grms
- Job recovery in progress. New job identifier is: 1085556984375_appid_1210
```

suspend_job

`grms-client` suspend_job *`jobId`*

Suspends whole job suspending all running tasks.

Important

This functionality is not implemented yet.

```
[piontek@druid-bis]$ grms-client suspend_job 1085141911671_appid_3916
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://druid-bis.man.poznan.pl:8442/axis/services/grms
- GRMS response was:
- Job suspending failed!
- errorCode = 156
- errorMessage = Not implemented yet
```

resume_job

`grms-client` resume_job *`jobId`* [*`jobDescription-file`*]

Resumes execution of previously suspended job.

> **Important**
>
> This functionality is not implemented yet.

```
[piontek@druid-bis]$ grms-client resume_job 1085141911671_appid_3916 ./exampleas/resume.xml
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://druid-bis.man.poznan.pl:8442/axis/services/grms
- Job resuming failed!
- errorCode = 156
- errorMessage = Not implemented yet
```

refresh_job

`grms-client` refresh_job *`jobId`*

Refreshes user proxy used to starts new tasks and also proxies for all running tasks.

```
[piontek@druid-bis]$ grms-client refresh_job 1085141911671_appid_3916
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://druid-bis.man.poznan.pl:8442/axis/services/grms
- Job refreshed successfully.
```

cancel_job

`grms-client` cancel_job *`jobId`*

Cancels job cancelling all running tasks.

```
 [piontek@druid-bis]$ grms-client cancel_job 1085141911671_appid_3916
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://druid-bis.man.poznan.pl:8442/axis/services/grms
- Job 1085141911671_appid_3916 has been cancelled successfully
```

If the current state of the job doesn't allow canceling it, the proper error message is displayed.

```
piontek@druid-bis]$ grms-client cancel_job 1085141911671_appid_3916
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: httpg://rage1.man.poznan.pl:8443/axis/services/grms
- Canceling of job failed!
- GRMS response was:
- errorCode: 313
- errorMessage: Job can not be canceled in current state
```

migrate_task

`grms-client` migrate_task *`jobId taskId`* [*`jobDescription-file`*]

Migrates task to a better resource if such one exists. It is possible to specify new job description that will be used to restart task on a new resource.

```
[piontek@druid-bis]$ grms-client migrate_task 1084456104272_appid_0728 taskid_1 \
                                                    ./examples/example2.xml
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://druid-bis.man.poznan.pl:8442/axis/services/grms
- Migration in progress. Invoke “grms-client task_info” to check the status of the operation.
```

If the migration cannot be performed because the current state of the task does not allow to migrate it, the proper error message is displayed.

```
[piontek@druid-bis]$ grms-client migrate_task 1085141911671_appid_3916 taskid_1 \
                                                    ./examples/example2.xml
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://druid-bis.man.poznan.pl:8442/axis/services/grms
- Job migration failed!
```

```
- GRMS response was:
- errorCode: 142
- errorMessage: Task in current state can not be migrated
```

suspend_task

`grms-client` suspend_task *`jobId taskId`*

Suspends execution of running task.

```
[piontek@druid-bis]$ grms-client suspend_task 1085141911671_appid_3916 taskid_1
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://druid-bis.man.poznan.pl:8442/axis/services/grms
- Suspending in progress.
  Invoke “grms-client task_info” to check the status of the operation.
```

resume_task

`grms-client` resume_task *`jobId taskId`* [*`jobDescription-file`*]

Resumes execution of previously suspended task. It is possible to specify new job description that will be used to "restart" task.

```
[piontek@druid-bis]$ grms-client resume_task 1085141911671_appid_3916 taskid_1 \
                                                          ./exampleas/resume.xml
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://druid-bis.man.poznan.pl:8442/axis/services/grms
- Resuming in progress.
  Invoke “grms-client task_info” to check the status of the operation.
```

cancel_task

`grms-client` cancel_task *`jobId taskId`*

Cancels given task.

```
 [piontek@druid-bis]$ grms-client cancel_task 1085141911671_appid_3916 taskid_1
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://druid-bis.man.poznan.pl:8442/axis/services/grms
- Task has been cancelled successfully
```

If the current state of the task doesn’t allow to cancel it, the proper error message is displayed.

```
piontek@druid-bis]$ grms-client cancel_job 1085141911671_appid_3916 taskid_1
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://druid-bis.man.poznan.pl:8442/axis/services/grms
- Canceling of job failed!
- GRMS response was:
- errorCode: 314
- errorMessage: Task can not be canceled in current state
```

commit_task

`grms-client` commit_job *`jobId taskId`*

Commits task submitted with two phase commit option (commitWait attribute)

```
[piontek@druid-bis]$ grms-client commit_task 1084456104272_appid_0728
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://druid-bis.man.poznan.pl:8442/axis/services/grms
- job committed successfully
```

If the job doesn't wait for commit, the proper error message is displayed.

```
[piontek@druid-bis]$ grms-client commit_job 1085141911671_appid_3916
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://druid-bis.man.poznan.pl:8442/axis/services/grms
- Job committing failed!
- GRMS response was:
- errorCode = 176
- errorMessage = Job already committed
```

list_all

`grms-client` list_all UNCOMMITED | SUBMITTED | ACTIVE | FINISHED | SUSPENDED | FAILED | CANCELED | BROKEN

Displays list of all jobs (regardless of the owner) in given state.

```
[piontek@druid-bis]$ grms-client list_all canceled
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://druid-bis.man.poznan.pl:8442/axis/services/grms
- List of jobs got successfully
- jobsList[0]=1084282948921_appid_8937
- jobsList[1]=1084360041571_appid_1513
- jobsList[2]=1085051139315_appid_7937
- jobsList[3]=1085054010026_appid_4949
- jobsList[4]=1085142249476_appid_2322
- jobsList[5]=1085167896526_appid_3682
- jobsList[6]=1086472467834_appid_4461
```

list_jobs

`grms-client` list_jobs [UNCOMMITED | SUBMITTED | ACTIVE | FINISHED | SUSPENDED | FAILED | CANCELED | BROKEN] [*limit*]

Returns list of jobs belonging to the user invoking the command. It is possible to specify optionally the status of job the user is interested in. In such case only jobs in given state will be listed additionally it is possible to limit list to some number of last submitted jobs.

```
[piontek@druid-bis]$ grms-client list_jobs canceled
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://druid-bis.man.poznan.pl:8442/axis/services/grms
- List of jobs got successfully
- jobsList[0]=1084282948921_appid_8937
- jobsList[1]=1084360041571_appid_1513
```

If the status is not specified all jobs belonging to the user are listed.

```
[piontek@druid-bis]$ grms-client list_jobs
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://druid-bis.man.poznan.pl:8442/axis/services/grms
- List of jobs got successfully
- jobsList[0]=1084282948921_appid_8937
- jobsList[1]=1084360041571_appid_1513
- jobsList[2]=1085051139315_appid_7937
- jobsList[3]=1085054010026_appid_4949
- jobsList[4]=1085142249476_appid_2322


[piontek@druid-bis]$ grms-client list_jobs 2
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://druid-bis.man.poznan.pl:8442/axis/services/grms
- List of jobs got successfully
- jobsList[0]=1084282948921_appid_8937
- jobsList[1]=1084360041571_appid_1513
```

list_tasks

`grms-client` list_tasks *jobId* [UNSUBMITTED | UNCOMMITTED | QUEUED | PREPROCESSING | PENDING | RUNNING | STOPPED | POSTPROCESSING | FINISHED | SUSPENDED | FAILED | CANCELED]

Lists tasks of given job. It is possible to limit list only to tasks in specified state.

```
[piontek@druid-bis]$ grms-client list_tasks failed
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://druid-bis.man.poznan.pl:8442/axis/services/grms
- List of tasks got successfully
- tasksList[0]=task_3
- tasksList[1]=task_5
```

If the status is not specified all tasks belonging to the job are listed.

```
[piontek@druid-bis]$ grms-client list_jobs
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
```

```
- Service URL: https://druid-bis.man.poznan.pl:8442/axis/services/grms
- List of tasks got successfully
- tasksList[0]=task_1
- tasksList[1]=task_2
- tasksList[2]=task_3
- tasksList[3]=task_4
- tasksList[4]=task_5
```

list_project

`grms-client` list_project [UNCOMMITED | SUBMITTED | ACTIVE | FINISHED | SUSPENDED | FAILED | CANCELED | BROKEN] [*`limit`*]

List jobs belonging to the given project. Optionally it is possible to specify state we are interested in and/or limit list to only some number of last jobs.

```
[piontek@druid-bis]$ grms-client list_project gridge_project
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://druid-bis.man.poznan.pl:8442/axis/services/grms
- List of jobs got successfully
- jobsList[0]=1084282948921_appid_8937
- jobsList[1]=1084360041571_appid_1513
- jobsList[2]=1085051139315_appid_7937
- jobsList[3]=1085054010026_appid_4949
- jobsList[4]=1085142249476_appid_2322


[piontek@druid-bis]$ grms-client list_project gridge_project active
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://druid-bis.man.poznan.pl:8442/axis/services/grms
- List of jobs got successfully
- jobsList[0]=1084282948921_appid_8937
- jobsList[1]=1084360041571_appid_1513


[piontek@druid-bis]$ grms-client list_project gridge_project active 1
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://druid-bis.man.poznan.pl:8442/axis/services/grms
- List of jobs got successfully
- jobsList[0]=1084282948921_appid_8937
```

register_access

`grms-client` register_access *`jobId taskId service-location pid`*

Registers information needed to checkpoint application using web service interface.

```
[piontek@druid-bis]$ grms-client register_access 1084367864146_appid_1408 task_1 \
                                  http://rage1.man.poznan.pl:4567 1234
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://druid-bis.man.poznan.pl:8442/axis/services/grms
- Task Application Access registered successfully
```

Important

In current version the correctness of the passed url is not checked.

get_access

`grms-client` get_access *`jobId taskId`*

Returns information registered by "*`register_access`*" command.

```
[piontek@druid-bis-bis]$ grms-client get_access 1084367864146_appid_1408 task_1
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://druid-bis.man.poznan.pl:8442/axis/services/grms
- Task Application Access got successfully
- ServiceLocation = http://rage1.man.poznan.pl:4567
- PID = 1234
```

unregister_access

`grms-client` unregister_access *`jobId taskId`*

Unregisters location of service implementing web service checkpoint interface.

```
[piontek@druid-bis]$ grms-client unregister_access 1084367864146_appid_1408 task_1
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://druid-bis.man.poznan.pl:8442/axis/services/grms
- Task Application Access unregistered successfully
```

job_info

`grms-client` job_info *`jobId`*

Returns complex information about the given job.

```
[piontek@druid-bis]$ grms-client info 1152877900969_example13_3744
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://druid-bis.man.poznan.pl:8442/axis/services/grms
- jobInfo[1152877900969_example13_3744] is:
ProjectId = Guide
UserDN = /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
Status = FINISHED
SubmissionTime = Fri Jul 14 13:51:41 CEST 2006
FinishTime = Fri Jul 14 14:02:34 CEST 2006
ErrorDescription =
tasksIdentifiers = migration
tasksCount = 1
waitForCommit = false
JobDescription =
<grmsJob appid="example13" project="Guide">
   <task taskid="migration">
      <resource>
         <hostname>fury.man.poznan.pl</hostname>
      </resource>
      <executable type="single" count="1" checkpointable="true">
         <execfile name="chkpt_test">
            <url>gsiftp://fury.man.poznan.pl/~/examples/chkpt_test</url>
         </execfile>
         <arguments>
            <value>6000</value>
         </arguments>
         <stdout>
            <url>${TASK_DIR}/stdout</url>
         </stdout>
         <stderr>
            <url>${TASK_DIR}/stderr</url>
         </stderr>
         <checkpoint>
            <file name="checkpoint" type="out">
               <url>gsiftp://fury.man.poznan.pl/~/examples/checkpoint-${JOB_ID}</url>
            </file>
            <file name="checkpoint" type="in">
               <url>gsiftp://fury.man.poznan.pl/~/examples/checkpoint-${JOB_ID}</url>
            </file>
         </checkpoint>
      </executable>
   </task>
</grmsJob>
```

task_info

`grms-client` task_info *`jobId taskId`* [*`limit`*]

Returns complex information about the given task. It is possible to limit information concerning history of task

```
[piontek@druid-bis]$ grms-client task_info 1152877900969_example13_3744 migration
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://druid-bis.man.poznan.pl:8442/axis/services/grms
TaskInformation[ 1152877900969_example13_3744,migration] =
Type = SINGLE
Status = FINISHED
SubmissionTime = Fri Jul 14 13:51:41 CEST 2006
FinishTime = Fri Jul 14 14:02:34 CEST 2006
ProxyLifetime = P0Y0M0D0H0M0S
RequestStatus = TASK_DONE
ErrorDescription =
waitForCommit = false
historyLength = 1
History:
History[ 0 ] =
StartTime = Fri Jul 14 13:51:41 CEST 2006
```

```
LocalSubmissionTime = Fri Jul 14 13:51:45 CEST 2006
LocalStartTime = Fri Jul 14 13:51:46 CEST 2006
LocalFinishTime = Fri Jul 14 14:02:34 CEST 2006
TaskDescription = <?xml version="1.0" encoding="UTF-8"?>
<task taskid="migration">
    <resource>
        <hostname>fury.man.poznan.pl</hostname>
    </resource>
    <executable type="single" count="1" checkpointable="true">
        <execfile name="chkpt_test">
            <url>gsiftp://fury.man.poznan.pl/~/examples/chkpt_test</url>
        </execfile>
        <arguments>
            <value>6000</value>
        </arguments>
        <checkpoint>
            <file name="checkpoint" type="out">
                <url>gsiftp://fury.man.poznan.pl/~/examples/checkpoint-${JOB_ID}</url>
            </file>
            <file name="checkpoint" type="in">
                <url>gsiftp://fury.man.poznan.pl/~/examples/checkpoint-${JOB_ID}</url>
            </file>
        </checkpoint>
    </executable>
</task>

No ApplicationAccess
Coallocation[ 0]=
HostName = fury.man.poznan.pl
Count = 0
Indexes =


[piontek@druid-bis]$ grms-client task_info 1152877900969_example13_3744 migration 0
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://druid-bis.man.poznan.pl:8342/axis/services/grms
- TaskInformation[ 1152877900969_example13_3744,migration] =
Type = SINGLE
Status = FINISHED
SubmissionTime = Fri Jul 14 13:51:41 CEST 2006
FinishTime = Fri Jul 14 14:02:34 CEST 2006
ProxyLifetime = P0Y0M0D0H0M0S
RequestStatus = TASK_DONE
ErrorDescription =
waitForCommit = false
historyLength = 1
History:
```

resources

`grms-client` resources *`taskId jobDescription-file`*

Returns list of resources that meet requirements specified for given task in job description.

```
[piontek@druid-bis]$ grms-client resources task_1 ./examples/example1.xml
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://druid-bis.man.poznan.pl:8442/axis/services/grms
- List of resources got successfully
- List of resources:
- eltoro.pcz.pl:2119/jobmanager-fork:/C=PL/O=GRID/O=Technical University of Czestochowa
     /CN=host/eltoro.pcz.pl
- packcs-e0.scai.fraunhofer.de:2119/jobmanager-fork:/C=DE/O=Fraunhofer/OU=SCAI
     /OU=Services/CN=packcs-e0.scai.fraunhofer.de
- bouscat.cs.cf.ac.uk:2119//jobmanager-fork
- onyx3.zib.de:2119/jobmanager:/O=Grid/O=GridLab/CN=onyx3.zib.de
- sr8000.lrz-muenchen.de:2119//jobmanager-fork
- helix.bcvc.lsu.edu:2119/jobmanager-fork:/O=Grid/O=GridLab/CN=helix.bcvc.lsu.edu
- peyote.aei.mpg.de:2119//jobmanager-fork
- litchi.zib.de:2119/jobmanager-fork:/O=Grid/O=GridLab/CN=litchi.zib.de
- n0.hpcc.sztaki.hu:2119//jobmanager-fork
- skirit.ics.muni.cz:2119/jobmanager-fork:/O=CESNET/O=Masaryk University
    /CN=host/skirit.ics.muni.cz
- hitcross.lrz-muenchen.de:2119//jobmanager-fork
- fs0.das2.cs.vu.nl:2119/jobmanager-fork:/O=dutchgrid/O=hosts/OU=cs.vu.nl/CN=fs0.das2.cs.vu.nl
```

test

`grms-client` test *`jobDescription-file`*

Checks the correctness of job description

```
[piontek@druid-bis]$ grms-client test ./examples/example2.xml
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://druid-bis.man.poznan.pl:8442/axis/services/grms
- Job description is correct


[piontek@druid-bis]$ grms-client test ./examples/error.xml
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://druid-bis.man.poznan.pl:8442/axis/services/grms
- Incorrect job description!
- GRMS response was:
- errorCode: 101
- errorMessage: Job description syntax error: The element type "value" must be terminated
                  by the matching end-tag "</value>".
```

add_job_notif

`grms-client` add_job_notif *`jobId`* SOAP *`destination`* [*`JOB_STATUS`* [`,` *`JOB_STATUS`*...]]

Registers SOAP notification concerning changes of job's status. It is possible to register only for some subset of statuses.

```
[piontek@druid-bis]$ grms-client add_job_notif 1107243045980_ps_extension_3879 soap \
                                 http://rage1.man.poznan.pl:1222
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://druid-bis.man.poznan.pl:8442/axis/services/grms
- Notification registered successfully.
  NotificationId = 1107243045980_ps_extension_3879:1107246112047


[piontek@druid-bis]$ grms-client add_job_notif 1107243045980_ps_extension_3879 soap \
                        http://rage1.man.poznan.pl:1222 active,finished
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://druid-bis.man.poznan.pl:8442/axis/services/grms
- Notification registered successfully.
  NotificationId = 1107243045980_ps_extension_3879:1107246123423
```

add_job_notif

`grms-client` add_job_notif *`jobId`* GASS *`destination`* TRUE | FALSE [*`format`* [*`JOB_STATUS`* [`,` *`JOB_STATUS`*...]]]

Registers GASS notification concerning changes of job's status. It is obligatory to specify jobId, location of remote file and choose if each notification should be appended to the file or should overwrite it. Optionally it is possible to specify format of message and list of statuses.

```
[piontek@druid-bis]$ grms-client add_job_notif 1107243045980_ps_extension_3879 gass \
                       http://fury.man.poznan.pl:55616/home/piontek/examples/notif.txt false
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://druid-bis.man.poznan.pl:8442/axis/services/grms
- Notification listener was registered successfully.
  notificationId = 1107243045980_ps_extension_3879:1107246124547


[piontek@druid-bis]$ grms-client add_job_notif 1107243045980_ps_extension_3879 gass \
                       http://fury.man.poznan.pl:55616/home/piontek/examples/notif.txt false \
                       "job %j changed status to %s!" active,finished
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://druid-bis.man.poznan.pl:8442/axis/services/grms
- Notification listener was registered successfully.
  notificationId = 1107243045980_ps_extension_3879:1107246112632
```

get_job_notif

`grms-client get_job_notif` *`jobId`*

Lists notifications registered for the given job.

```
[piontek@druid-bis] ~/GRMS-client $ grms-client get_job_notif 1153129223646_append_8646
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://druid-bis.man.poznan.pl:8342/axis/services/grms
- List of notifications got successfully.
- JobStatusNotification[0]=
NotificationId = 1153129223646_append_8646:1153291880082
Protocol = GASS
Destination = http://fury.man.poznan.pl:2222
HasMask = true
```

```
AppendMode = true
Format = Job %j changed state to %s
Mask = CANCELED FAILED
- JobStatusNotification[1]=
NotificationId = 1153129223646_append_8646:1153291809189
Protocol = SOAP
Destination = http://fury.man.poznan.pl:12345
HasMask = true
Mask = FINISHED ACTIVE
```

get_job_notif

`grms-client get_job_notif` *`jobId notificationId`*

Returns description of notification with given identifier.

```
piontek@druid-bis]$ grms-client get_job_notif 1153129223646_append_8646 \
                                              1153129223646_append_8646:1153291809189
GRMS_URL [https://fury.man.poznan.pl:8342/axis/services/grms]
GRMS_DN [/C=PL/O=GRID/O=PSNC/CN=grms/fury.man.poznan.pl]
GRMS_DELEG_TYPE [FULL]
GRMS_TIMEOUT [5]
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://fury.man.poznan.pl:8342/axis/services/grms
- NotificationInformation:
NotificationId = 1153129223646_append_8646:1153291809189
Protocol = SOAP
Destination = http://fury.man.poznan.pl:12345
HasMask = true
Mask = FINISHED ACTIVE
```

If the wrong NotificationId was used the proper error message is displayed.

```
[piontek@druid-bis]$ grms-client get_job_notif 1107243045980_ps_ext_3879 wrong_notif_id
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://druid-bis.man.poznan.pl:8442/axis/services/grms
- Getting Notification info failed!
- GRMS response was:
- errorCode: 165
- errorMessage: Failed to get notification - no such notification
```

del_job_notif

`grms-client del_job_notif` *`jobId notificationId`*

Unregisters notification with given identifier

```
[piontek@druid-bis]$ grms-client del_job_notif 1107243045980_ps_ext_3879 \
                                               1107243045980_ps_ext_3879:1107246248349
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://druid-bis.man.poznan.pl:8442/axis/services/grms
- Notification unregistered successfully.
```

add_task_notif

`grms-client` add_task_notif *`jobId taskId`* STATUS SOAP *`destination`* [*`TASK_SATUS`* [*`,TASK_STATUS`*...]]

For the given task registers SOAP notification concerning changes of task's status. It is possible to specify subset of statuses.

```
[piontek@druid-bis]$ grms-client add_task_notif 1153129223646_append_8646 date status soap \
                                      https://fury.man.poznan.pl:3333 running,finished,canceled
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://fury.man.poznan.pl:8342/axis/services/grms
- Notification registered successfully. NotificationId = 1153129223646_append_8646:1153296496709
```

add_task_notif

`grms-client` add_task_notif *`jobId taskId`* STATUS GASS *`destination`* TRUE | FALSE [*`format`* [*`TASK_STATUS`* [*`,TASK_STATUS`*...]]]

For the given task registers GASS notification concerning changes of task's status. It is possible to specify subset of statuses.

```
piontek@druid-bis]$ grms-client add_task_notif 1153129223646_append_8646 date status gass \
                                https://fury.man.poznan.pl:3333 true \
                                "Task %j,%t changed status to %s" \
                                running,finished,canceled
GRMS_URL [https://fury.man.poznan.pl:8342/axis/services/grms]
GRMS_DN [/C=PL/O=GRID/O=PSNC/CN=grms/fury.man.poznan.pl]
GRMS_DELEG_TYPE [FULL]
GRMS_TIMEOUT [5]
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://fury.man.poznan.pl:8342/axis/services/grms
- Notification registered successfully. NotificationId = 1153129223646_append_8646:1153296965301
```

add_task_notif

grms-client add_task_notif *jobId taskId* REQUEST SOAP *destination* [*TASK_REQUEST_SATUS* [,*TASK_REQUEST_STATUS*...]]

For the given task registers SOAP notification concerning changes of request processing. It is possible to specify subset of statuses.

```
[piontek@druid-bis]$ grms-client add_task_notif 1153129223646_append_8646 date request soap \
                              https://fury.man.poznan.pl:1234 task_resource,task_exec_active
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://fury.man.poznan.pl:8342/axis/services/grms
- Notification registered successfully.
  NotificationId = 1153129223646_append_8646:1153296497634
```

add_task_notif

grms-client add_task_notif *jobId taskId* REQUEST GASS *destination* TRUE | FALSE [*format* [*TASK_REQUEST_STATUS* [,*TASK_REQUEST_STATUS*...]]]

For the given task registers GASS notification concerning changes of request processing. It is possible to specify subset of statuses.

```
[piontek@druid-bis]$ grms-client add_task_notif 1153129223646_append_8646 date request gass \
                                https://fury.man.poznan.pl:1236 false \
                                "Task %j,%t changed status to %s" task_resource,task_exec_active
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://fury.man.poznan.pl:8342/axis/services/grms
- Notification registered successfully.
  NotificationId = 1153129223646_append_8646:1153296503562
```

add_tasks_notif

grms-client add_tasks_notif *jobId* STATUS SOAP *destination* [*TASK_STATUS* [,*TASK_STATUS*...]]

For all tasks of given job registers SOAP notifications concerning changes of statuses of tasks. It is possible to specify subset of statuses.

```
[piontek@druid-bis]$ grms-client add_tasks_notif 1153129223646_append_8646 status soap \
                                   https://fury.man.poznan.pl:3333 running,finished,canceled
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://fury.man.poznan.pl:8342/axis/services/grms
- Notifications registered successfully
```

add_tasks_notif

grms-client add_tasks_notif *jobId* STATUS GASS *destination* TRUE | FALSE [*format* [*TASK_STATUS* [,*TASK_STATUS*...]]]

For all tasks of given job registers GASS notifications concerning changes of statuses of tasks. It is possible to specify if notifications should be appended to the file or overwrite it, format of message and subset of statuses.

```
piontek@druid-bis]$ grms-client add_tasks_notif 1153129223646_append_8646 status gass \
                                             https://fury.man.poznan.pl:3333 true \
                                             "Task %j,%t changed status to %s" \
                                             running,finished,canceled
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://fury.man.poznan.pl:8342/axis/services/grms
- Notifications registered successfully
```

add_tasks_notif

`grms-client` add_tasks_notif *`jobId`* REQUEST SOAP *`destination`* [*`TASK_REQUEST_STATUS`* [*`,TASK_REQUEST_STATUS`*...]]

For all tasks of given job registers SOAP notifications concerning processing of requests. It is possible to specify subset of statuses.

```
[piontek@druid-bis]$ grms-client add_tasks_notif 1153129223646_append_8646 request soap \
                              https://fury.man.poznan.pl:1234 task_resource,task_exec_active
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://fury.man.poznan.pl:8342/axis/services/grms
- Notifications registered successfully
```

add_tasks_notif

`grms-client` add_tasks_notif *`jobId`* REQUEST GASS *`destination`* TRUE | FALSE [*`format`* [*`TASK_REQUEST_STATUS`* [*`,TASK_REQUEST_STATUS`*...]]]

For all tasks of given job registers GASS notifications concerning processing of requests. It is possible to specify if notifications should be appended to the file or overwrite it, format of message and subset of statuses.

```
[piontek@druid-bis]$ grms-client add_tasks_notif 1153129223646_append_8646 request gass \
                              https://fury.man.poznan.pl:1236 false \
                              "Task %j,%t changed status to %s" task_resource,task_exec_active
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://fury.man.poznan.pl:8342/axis/services/grms
- Notifications registered successfully
```

get_task_notif

`grms-client info_task_notif` *`jobId taskId`* [STATUS | REQUEST]

Returns descriptions of notifications registered for given task. Optionally it is possible to specify type of notifications we are interested in.

```
piontek@druid-bis]$ grms-client get_task_notif 1153129223646_append_8646 date
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://druid-bis.man.poznan.pl:8342/axis/services/grms
- List of notifications got successfully.
- notification[0]=
NotificationId = 1153129223646_append_8646:1153309627232
Protocol = GASS
Destination = https://fury.man.poznan.pl:3333
HasMask = true
AppendMode = true
Format = Task %j,%t changed status to %s
Event = REQUEST
- notification[1]=
NotificationId = 1153129223646_append_8646:1153307794500
Protocol = GASS
Destination = https://fury.man.poznan.pl:3333
HasMask = true
AppendMode = true
Format = Task %j,%t changed status to %s
Event = STATUS
- notification[2]=
NotificationId = 1153129223646_append_8646:1153296965301
Protocol = GASS
Destination = https://fury.man.poznan.pl:3333
HasMask = true
AppendMode = true
Format = Task %j,%t changed status to %s
Event = STATUS
- notification[3]=
NotificationId = 1153129223646_append_8646:1153296496709
Protocol = SOAP
Destination = https://fury.man.poznan.pl:3333
HasMask = true
Event = STATUS


piontek@druid-bis]$ grms-client get_task_notif 1153129223646_append_8646 date request
GRMS_URL [https://fury.man.poznan.pl:8342/axis/services/grms]
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://fury.man.poznan.pl:8342/axis/services/grms
- List of notifications got successfully.
```

```
- notification[0]=
NotificationId = 1153129223646_append_8646:1153309627232
Protocol = GASS
Destination = https://fury.man.poznan.pl:3333
HasMask = true
AppendMode = true
Format = Task %j,%t changed status to %s
Event = REQUEST
```

get_task_notif

`grms-client get_task_notif` *`jobId taskId notificationId`* [STATUS | REQUEST]

Returns description of notification with given identifier. Optionally it is possible to specify type of notification.

```
piontek@druid-bis]$ grms-client get_task_notif 1153129223646_append_8646 date \
                                                1153129223646_append_8646:1153309627232
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://druid-bis.man.poznan.pl:8342/axis/services/grms
- NotificationInformation:
NotificationId = 1153129223646_append_8646:1153309627232
Protocol = GASS
Destination = https://fury.man.poznan.pl:3333
HasMask = true
AppendMode = true
Format = Task %j,%t changed status to %s
Event = REQUEST


piontek@druid-bis]$ grms-client get_task_notif 1153129223646_append_8646 date \
                                          1153129223646_append_8646:1153309627232 request
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://druid-bis.man.poznan.pl:8342/axis/services/grms
- NotificationInformation:
NotificationId = 1153129223646_append_8646:1153309627232
Protocol = GASS
Destination = https://fury.man.poznan.pl:3333
HasMask = true
AppendMode = true
Format = Task %j,%t changed status to %s
Event = REQUEST
Mask = TASK_UNCOMMITTED
```

If the wrong type is specified for notification, the proper error message will be displayed

```
piontek@druid-bis ~/GRMS-client $ grms-client get_task_notif 1153129223646_append_8646 \
                                    date 1153129223646_append_8646:1153309627232 status
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://fury.man.poznan.pl:8342/axis/services/grms
- Getting notification information failed!
- grms.services.gridge.grms.exceptions.GetTaskStatusNotificationException
errorCode = 165
errorMessage = Failed to get notification - no such notification
```

del_task_notif

`grms-client del_task_notif` *`jobId taskId notificationId`*

For a given task unregisters notification with given identifier.

```
[piontek@druid-bis]$ grms-client del_task_notif 1153129223646_append_8646 date \
                                                  1153129223646_append_8646:1153309627232
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://fury.man.poznan.pl:8342/axis/services/grms
- Notification unregistered successfully.
```

add_file_dir

`grms-client` add_file_dir *`jobId taskId overwrite FILE_DIR_LOGICAL_TYPE name`* PHYSICAL | LOGICAL *`path user`* FILE | DIRECTORY *`append required`* [*`permissions`*]

Dynamically adds file or directory for a given task. It is possible to add argument as well as checkpoint files and directories, specify if in case of presence of such file it should be overwritten or cause failure. Additionally user can specify if during the transfer the contents of file/directory should be appended or overwrite the destination, if the file or directory is required and if its lack should cause failure. Optionally it is possible to specify permis-

sions.

```
[piontek@druid-bis]$ grms-client add_file_dir 1153129223646_append_8646 date \
                                        false argument_in file_name physical \
                                        gsiftp://fury.man.poznan.pl/~/examples/input.txt "" \
                                        file false true 766
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://fury.man.poznan.pl:8342/axis/services/grms
- Operation finished successfully
```

If the file or directory to be registered already exists and "*overwrite*" parameter is set to "false" the proper error message is displayed.

```
[piontek@druid-bis]$ grms-client add_file_dir 1153129223646_append_8646 date \
                                        false argument_in file_name physical \
                                        gsiftp://fury.man.poznan.pl/~/examples/input.txt "" \
                                        file false true 766
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://fury.man.poznan.pl:8342/axis/services/grms
- Adding file/dir failed!
- Number of errors: 1
Result[0] errorCode=420 errorMessage=File/Dir already exists
```

If the file or directory definition is incorrect the proper error message is displayed.

```
piontek@druid-bis ~/GRMS-client $ grms-client add_file_dir 1153129223646_append_8646 date \
                                       false argument_in "" physical \
                                       gsiftp://fury.man.poznan.pl/~/examples/input.txt "" \
                                       file false true 766
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://fury.man.poznan.pl:8342/axis/services/grms
- Adding output file/dir failed!
- Number of errors: 1
Result[0] errorCode=422 errorMessage=Incorrect definition
```

get_file_dirs

grms-client get_file_dirs *jobId taskId* [*FILE_DIR_LOGICAL_TYPE* [,*FILE_DIR_LOGICAL_TYPE*...]] [*FILE_DIR_ORIGIN_TYPE* [,*FILE_DIR_ORIGIN_TYPE*...]]

For the given task returns list of files and directories. Optionally it is possible to specify list of logical types and origins.

```
[piontek@druid-bis]$ grms-client get_file_dirs 1153392145921_example14_3397 tar
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://fury.man.poznan.pl:8342/axis/services/grms
- List of files/directories got successfully
- FileDir[0]:
- LogicalType = ARGUMENT_OUT
Name = ps.out.tgz
Path = gsiftp://fury.man.poznan.pl/~/examples/ps-${JOB_ID}.tgz
Append = false
Permissions = NOT SET
Required = true
StoreType = PHYSICAL
PhysicalType = FILE
OriginType = DESCRIPTION
- FileDir[1]:
- LogicalType = ARGUMENT_IN
Name = ps_in
Path = gsiftp://fury.man.poznan.pl/~/ps-${JOB_ID}.tgz
Append = true
Permissions = rw-rw-rw-
Required = false
StoreType = PHYSICAL
PhysicalType = FILE
OriginType = DESCRIPTION
- FileDir[2]:
- LogicalType = ARGUMENT_IN
Name = file_name
Path = 12345
User = GRMS
Append = false
Permissions = rw-r--r--
Required = true
StoreType = LOGICAL
PhysicalType = FILE
```

```
OriginType = ADDED


[piontek@druid-bis]$ grms-client get_file_dirs 1153392145921_example14_3397 tar argument_in \
                                                                          description,added
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://fury.man.poznan.pl:8342/axis/services/grms
- List of files/directories got successfully
- FileDir[0]:
- LogicalType = ARGUMENT_IN
Name = ps_in
Path = gsiftp://fury.man.poznan.pl/~/ps-${JOB_ID}.tgz
Append = true
Permissions = rw-rw-rw-
Required = false
StoreType = PHYSICAL
PhysicalType = FILE
OriginType = DESCRIPTION
- FileDir[1]:
- LogicalType = ARGUMENT_IN
Name = file_name
Path = 12345
User = GRMS
Append = false
Permissions = rw-r--r--
Required = true
StoreType = LOGICAL
PhysicalType = FILE
OriginType = ADDED


[piontek@druid-bis]$ grms-client get_file_dirs 1153392145921_example14_3397 tar argument_in added
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://fury.man.poznan.pl:8342/axis/services/grms
- List of files/directories got successfully
- FileDir[0]:
- LogicalType = ARGUMENT_IN
Name = file_name
Path = 12345
User = GRMS
Append = false
Permissions = rw-r--r--
Required = true
StoreType = LOGICAL
PhysicalType = FILE
OriginType = ADDED
```

del_file_dir

grms-client del_file_dir *jobId taskId FILE_DIR_LOGICAL_TYPE name* PHYSICAL | LOGICAL *path user* FILE | DIRECTORY

Unregisters file or directory.

```
[piontek@druid-bis]$ grms-client del_file_dir 1153392145921_example14_3397 tar argument_in \
                                                    file_name logical 12345 "GRMS" file
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://druid-bis.man.poznan.pl:8342/axis/services/grms
- Operation finished successfully
```

If the file or directory to be unregistered doesn't exist the proper error message is displayed.

```
[piontek@druid-bis]$ grms-client del_output 1088499660886_appid_1622 taskid_1 param_file \
                      gsiftp://rage1.man.poznan.pl/~/examples/param_file physical file
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://druid-bis.man.poznan.pl:8442/axis/services/grms
- The operation finished with result:
- errorCode= 421
- errorMessage= No such file or directory
```

del_file_dirs

grms-client del_file_dirs *jobId taskId* [*FILE_DIR_LOGICAL_TYPE* [,*FILE_DIR_LOGICAL_TYPE*...]] [*FILE_DIR_ORIGIN_TYPE* [,*FILE_DIR_ORIGIN_TYPE*...]]

Unregisters files or directories of specific type.

```
[piontek@druid-bis]$ grms-client del_file_dirs 1153392145921_example14_3397 tar argument_in \
```

```
                                                                    description,added
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://fury.man.poznan.pl:8342/axis/services/grms
- Operation finished successfully
```

extend_execution

`grms-client` extend_execution *`jobId taskId`* +|- *`years months days hours minutes seconds`*

For a time scheduled tasks extends its time of execution.

```
[piontek@druid-bis]$ grms-client extend_execution 1088499660886_appid_1622 taskid_1 + 0 0 0 2 30 0
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://druid-bis.man.poznan.pl:8443/axis/services/grms
- ExecutionTime extended successfully
```

description

`grms-client` description SHORT | FULL

Returns description of service.

```
[piontek@druid-bis]$ grms-client description short
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://druid-bis.man.poznan.pl:8442/axis/services/grms
- Gridge Resource Management System (GRMS) 1.0


[piontek@druid-bis]$ grms-client description full
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: https://druid-bis.man.poznan.pl:8442/axis/services/grms
- Gridge Resource Management System (GRMS) 1.0

Service URL: https://fury.man.poznan.pl:8342/axis/services/grms
Client host name:  druid-bis.man.poznan.pl
Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
Your Proxy: WAS DELEGATED SUCCESSFULLY

For more details please see: www.gridge.org
```