



# GRMS User Guide v. 2.0

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>

## Table of Contents

Introduction .....	1
GRMS functionality .....	2
Job submission and control .....	2
Task submission and control .....	3
Listing jobs according to some criteria .....	3
Listing tasks according to some criteria .....	4
Managing tasks .....	5
Getting information about the job .....	5
Getting information about the task .....	5
Finding resources .....	6
Notifications .....	6
Auxiliary functionality .....	7
GRMS interface .....	7
GRMS Job Description (GJD) .....	16
Brief Specification of GJD .....	17
JobDescription examples .....	24
GRMS java client .....	30
Requirements. ....	30
Installation and configuration of GRMS client .....	31
Usage of GRMS commandline client .....	31

## Introduction

The main aim of this guide is to introduce functionalities provided by GRMS v 2.0. In a nutshell, GridLab Resource Management System (GRMS) is an open source meta-scheduling system, developed under the GridLab IST-2001-32133 project ([www.gridlab.org](http://www.gridlab.org) [???]), 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 a 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 submissions to various batch queuing systems, clusters or resources directly. It has been designed as an independent set of core components for resource management processes, which can take advantage of various low-level Core Services and existing technologies (please note that we consider services taken from Globus Toolkit 2.X (Globus Pre WS) as Core Services, namely GRAM, gridFTP or MDS). Finally, GRMS can be considered as a robust system, which provides an abstraction of the complex grid infrastructure as well as a toolbox, which helps to form and adapt to distributing computing environments.

GRMS is based on Globus Toolkit 2.X and uses its services deployed on resources. It means that GRMS provides job and resource management mechanisms on the top of Core Services. Moreover, GRMS supports Grid Security Infrastructure by providing GSI-enabled Web Service interfaces for all clients, e.g. portals, command line clients or applications. Therefore, it can be integrated with other service-oriented grid middleware infrastructures. GRMS has been developed entirely in Java, and thus could be installed on many different kinds of op-

erating systems and resources. One of the main assumption 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 through XML-based resource specification documents, called GRMS Job Description, and sent to GRMS as SOAP requests over GSI transport layer connections.

Simultaneously, Resource Administrators (Resource Owners) have a 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.

## GRMS functionality

GRMS capabilities provided for end users can be divided into several groups according to GRMS's functionality. The following groups can be distinguished:

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

The next sections will describe all these functionality groups in more technical details.

## Job submission and control

The most important part of the GRMS's functionality is the ability to submit a job. Grms job consist of one or many tasks that can be dependent from each other. Each task can be executed by GRMS only if all tasks it depends on are in specified state. Each task is submitted to "the best" resource according to task resource requirements. In general, there are two possibilities: a user can specify "the best" resource in advance or "the best" resource will be chosen by GRMS. If the user specifies the resource in advance no scheduling and matching techniques will be used. Otherwise, GRMS will use a multi-criteria matchmaking algorithm to choose "the best" resource for the task. For job submission and control the XML-based document called GRMS Job Description has to be used by all clients. Using this document the user passes to GRMS all information needed by the system to execute tasks to be a parts of the job (for example, the location of the executables, files that have to be staged in, arguments, environment variables, checkpoint files, etc.) and all dependencies between tasks. The GRMS Job Description schema and examples will be described and explained in one of next chapters.

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

- `submitJob` – it is the main functionality of GRMS. Using it the user can submit the job (set of tasks) described in a 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.



- `migrateJob` – this functionality allows the user to migrate the job (all running tasks) to “better” resources (if such resources exist). The job is identified by the job identifier returned by the `submitJob` method. User can specify new job description. If new description is not specified, the original one will be used.
- `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 use the previous one.
- `cancelJob` – this functionality allows the user to stop the running job. The difference between the `suspendJob` and `cancelJob` functionality is that all tasks are stopped (killed) by the system and checkpoint operation is not performed for running tasks.

## 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:

- `submitTask` – Using it user can submit one task described in a GRMS Job Description. If the description is valid and task doesn't depends on other ones GRMS returns a globally unique job identifier (`GRMS_JOB_ID`), which unambiguously identifies the job in the system.
- `migrateTask` – this functionality allows the user to migrate one task to a “better” resource (if such resource exist). The task is identified by the job identifier returned by the `submitJob` or `submitTask` methods and task identifier specified by the user in job description. The whole process of task migration is relatively simple. First, the task is checkpointed on the resource, which is currently running on and then restarted on a new resource pointed by the user or chosen by GRMS. Note, that 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 would be possible only when a better resource was found. If the task is checkpointable (to be checkpointable the task has to either implement "checkpoint" web service interface and be registered in GRMS or be able in proper way to serve GAT checkpoint command sent by Mercury Service) GRMS perform checkpoint operation. If the task is not checkpointable it is killed by GRMS and the latest periodic checkpoint file is used.
- `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 task which was previously suspended. It is possible to define a new job description or use the previous one.
- `cancelTask` – this functionality allows the user to stop 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.

## Listing jobs according to some criteria

GRMS is able to return a list of jobs (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 all jobs in given state in the system. The following methods provide the aforementioned functionality:

- `getJobsList` – returns a list of jobs belonging to the user, optionally it is possible to define the requested sta-



tus,

- `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,

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

- `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 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.

## Listing tasks according to some 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,
- `QUEUED` – the task was put the queue and waits for execution,
- `PREPROCESSING` – GRMS make 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 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 Mercury checkpoint command and GAT mechanisms.
- `unregisterTaskApplicationAccess` - this functionality gives the possibility to unregister the job for checkpointing,
- `getTaskApplicationAccess` - this functionality allows to get information about settings related to the "checkpoint" functionality,
- `addTaskOutputFileDirs`, `addTaskCheckpointFileDirs` – 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.
- `getTaskOutputFileDirs`, `getTaskCheckpointFileDirs` – this functionality returns a list of output/checkpoint files and directories registered for the given task,
- `deleteTaskOutputFileDirs`, `deleteTaskCheckpointFileDirs` – this 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 given identifier.

- `getJobInformation` – this functionality returns the general information about the job (userDN, project, submission time, status, finish time, state description, list of tasks, job description),

For more information please see the proper paragraph of “GRMS Interface” chapter.

## Getting information about the task

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

- `getTaskInformation` – this functionality returns the general information about the task (submission time, status, finish time, state description, history length and the last history item),
- `getTaskHistory` - returns the array of information connected with the history of the task (task description, local submission time, local start time, local finish time, applicationAccess, etc.).



For more information please see the proper paragraph of “GRMS Interface” chapter.

## 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

User can register for notifications concerning the whole job or single tasks. 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”. Notifications mechanism was for example used to build more advanced “Notification Services” namely “Gridlab Notification Service” that can “forward” notification sent by GRMS to a group of users as e-mail or SMS.

Currently GRMS is able to send notification in two ways: using SOAP protocol and writing to a remote file. In case of SOAP protocol service registered for notifications has to implement `GrmsNotification` API described by `GrmsNotification.wsdl` file. Registering remote file as a destination of notifications user can specify the format of message (text containing list and order of details) and specify if next notifications should be appended to the file or overwrite its content.

`GRMSNotification` contains following information:

- `jobId [j]` - GRMS job identifier,
- `taskId [t]` - GRMS task identifier,
- `notificationId [n]` - notification identifier
- `eventType [e]` - type of the event (STATUS or REQUEST)
- `time [c or C]` - time when the event occurred. It is possible to specify if the time should be in human readable format "January 25, 2005, 17:31:10 GMT" or as a number of milliseconds since January 1, 1970, 00:00:00 GMT.
- `user [u]` - user whom the job or task belongs to
- `project [p]` - project which the job or task belongs to,
- `status [s]` - status of the job or task,
- `errorDescription [d]` - message describing the cause of last error.

## Important

The letter in square brackets after the name of parameter is a mark that represents following information in string describing format of the notification message. The format string can contain any text in which set of defined above marks preceded by “%” will be replaced by information taken from `GrmsNotification`. For example "Job %j has changed its status to %s at %c !".



Following set of methods can be used to manage GRMS notifications:

- `registerJobNotification` - functionality allows to register location of service for notifications concerning change 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)
- `unregisterJobNotification` - allows the user to unregister the notification.
- `getJobNotificationsList` - returns list of notifications registered for the given job,
- `getJobNotificationInformation` - returns details concerning one of the registered notifications (protocol, destination, appendMode, format).
- `registerTaskNotification` - functionality allows to register location of service for requested type of events (status or request). GRMS is able to send notification every time when the status of task has changed (for example from PENDING to ACTIVE) or when the grms goes to the next step performing user's request (for example when staging files has finished).
- `unregisterTaskNotification` - allows the user to unregister the notification.
- `getTaskNotificationsList` - returns list of notifications registered for the given task,
- `getTaskNotificationInformation` - returns details concerning one of the registered notifications (eventType, protocol, destination, appendMode, format).

## 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.

## GRMS interface

GRMS 2.0 provides the following meta-interface. All GRMS methods in case of execution failure "throws" an appropriate fault (exception) containing `errorCode` and `errorMessage` describing the cause of error. Additionally faults concerning jobs and tasks contain information about job and task identifiers, what gives possibility to work with grms-service in more flexible way.

### Definitions of faults.

```
GrmsException {
    int errorCode;
    String errorMessage;
}

GrmsJobException extends GrmsException {
    GrmsJobIdentifier jobIdentifier;
}
```



```
GrmsTaskException extends GrmsJobException {
    GrmsTaskIdentifier taskIdentifier;
}
```

```
GrmsProjectException extends GrmsException {
    GrmsProjectIdentifier projectIdentifier;
}
```

- GrmsException - base class for all Grms faults. Contains numerical error code and error description.
- GrmsJobException - extends GrmsException adding GrmsJobIdentifier.
- GrmsTaskException - extends GrmsJobException adding GrmsTaskIdentifier.
- GrmsProjectException - extends GrmsException adding GrmsProjectIdentifier.

### **submitJob.**

```
GrmsJobIdentifier submitJob( GrmsJobDescription) \
    throws GrmsSubmitJobException;
```

```
GrmsSubmitJob extends GrmsException;
```

Submits to GRMS the job described by GrmsJobDescription; returns a GrmsJobIdentifier.

### **migrateJob.**

```
void migrateJob( GrmsJobIdentifier, GrmsJobDescription) \
    throws GrmsMigrateJobException;
```

```
void migrateJob( GrmsJobIdentifier) \
    throws GrmsMigrateJobException;
```

```
GrmsMigrateJobException extends GrmsJobException;
```

Performs migration of the job with given identifier, according to provided job description or the previous one if a new description was not passed.

### **suspendJob.**

```
void suspendJob( GrmsJobIdentifier, GrmsJobDescription) \
    throws GrmsSuspendJobException;
```

```
void suspendJob( GrmsJobIdentifier) \
    throws GrmsSuspendJobException;
```

```
GrmsSuspendJobException extends GrmsJobException;
```

Suspends the execution of the job with given identifier, according to the provided job description or the previous one if a new description was not passed.

### **resumeJob.**

```
void resumeJob( GrmsJobIdentifier, GrmsJobDescription) \
    throws GrmsResumeJobException;
```

```
void resumeJob( GrmsJobIdentifier) \
    throws GrmsResumeJobException;
```

```
GrmsResumeJobException extends GrmsJobException;
```

Resumes the execution of the job with given identifier, according to the provided job description or the previous one if a new description was not passed.

### **submitTask.**

```
GrmsJobIdentifier submitJob( GrmsTaskIdentifier, GrmsJobDescription) \
    throws GrmsSubmitTaskException;
```



```
GrmsSubmitTaskException extends GrmsTaskException;
```

Submits to the task with given identifier described by GrmsJobDescription; returns a GrmsJobIdentifier.

#### **migratTask.**

```
void migrateTask( GrmsJobIdentifier, GrmsTaskIdentifier, GrmsJobDescription) \  
    throws GrmsMigrateTaskException;
```

```
void migrateTask( GrmsJobIdentifier, GrmsTaskIdentifier) \  
    throws GrmsMigrateTaskException;
```

```
GrmsMigrateTaskException extends GrmsTaskException;
```

Performs migration of the given task, according to the provided job description or the previous one if a new description was not passed.

#### **suspendTask.**

```
void suspendTask( GrmsJobIdentifier, GrmsTaskIdentifier, GrmsJobDescription) \  
    throws GrmsSuspendTaskException;
```

```
void suspendTask( GrmsJobIdentifier, GrmsTaskIdentifier) \  
    throws GrmsSuspendTaskException;
```

```
GrmsSuspendTaskException extends GrmsTaskException;
```

Suspends the execution of the given task, according to the provided job description or the previous one if a new description was not passed.

#### **resumeTask.**

```
void resumeTask( GrmsJobIdentifier, GrmsTaskIdentifier, GrmsJobDescription) \  
    throws GrmsResumeTaskException;
```

```
void resumeTask( GrmsJobIdentifier, GrmsTaskIdentifier) \  
    throws GrmsResumeTaskException;
```

```
GrmsResumeTaskException extends GrmsTaskException;
```

Resumes the execution of the job with given identifier, according to the provided job description or the previous one if a new description was not passed.

#### **cancelJob.**

```
void cancelJob( GrmsJobIdentifier) \  
    throws GrmsCancelJobException;
```

```
GrmsCancelJobException extends GrmsJobException;
```

Cancels the execution of the job with the given identifier.

#### **cancelTask.**

```
void cancelTask( GrmsJobIdentifier, GrmsTaskIdentifier) \  
    throws GrmsCancelTaskException;
```

```
GrmsCancelTaskException extends GrmsTaskException;
```

Cancels the execution of the given task.

#### **getAllJobsList.**

```
GrmsJobIdentifier[] getAllJobsList( GrmsJobStatus) \  
    throws GrmsGetAllJobsListException;
```

```
GrmsGetAllJobsListException extends GrmsException;
```

Returns a list of all jobs in the given state.

#### **getJobsList.**



```
GrmsJobIdentifier[] getJobsList() \
    throws GrmsGetJobsListException;

GrmsJobIdentifier[] getJobsList( GrmsJobStatus) \
    throws GrmsGetJobsListException;

GrmsGetJobsListException extends GrmsException;
```

Returns a list of all jobs or jobs in given state that belong to the user who invoked this method.

### **getTasksList.**

```
GrmsTaskIdentifier[] getTaskList( GrmsJobIdentifier) \
    throws GrmsGetTasksListException;

GrmsTaskIdentifier[] getTaskList( GrmsJobIdentifier, GrmsTaskStatus) \
    throws GrmsGetTasksListException;

GrmsGetTasksList extends GrmsJobException;
```

Returns a list of all tasks or tasks in given state that belong to the given job.

### **getProjectJobsList.**

```
GrmsJobIdentifier[] getProjectJobsList( GrmsProjectIdentifier) \
    throws GrmsGetProjectJobsListException;

GrmsJobIdentifier[] getProjectJobsList( GrmsProjectIdentifier, GrmsJobStatus) \
    throws GrmsGetProjectJobsListException;

GrmsGetProjectJobsList extends GrmsProjectException;
```

Returns a list of all jobs or jobs in given state belonging to the specified project.

### **registerTaskApplicationAccess.**

```
void registerTaskApplicationAccess( GrmsJobIdentifier, GrmsTaskIdentifier, GrmsTaskApplicationAccess) \
    throws GrmsRegisterTaskApplicationAccessException;

GrmsRegisterTaskApplicationAccess extends GrmsTaskException;

GrmsTaskApplicationAccess {
    String location;
    int pid;
}
```

Registers callback access for running application.

### **unregisterTaskApplicationAccess.**

```
void unregisterTaskApplicationAccess( GrmsJobIdentifier, GrmsTaskIdentifier) \
    throws GrmsUnregisterTaskApplicationAccessException;

GrmsUnregisterTaskApplicationAccess extends GrmsTaskException;
```

Unregisters callback access for running application.

### **getTaskApplicationAccess.**

```
GrmsTaskApplicationAccess getTaskApplicationAccess( GrmsJobIdentifier, GrmsTaskIdentifier) \
    throws GrmsGetTaskApplicationAccessException;

GrmsGetTaskApplicationAccessException extends GrmsTaskException;
```

Returns information about registered callback to running application (location of registered checkpoint interface and process identifier).

### **getJobInformation.**

```
GrmsJobInformation getJobInformation( GrmsJobIdentifier) \
    throws GrmsGetJobInformationException;

GrmsGetJobInformationException extends GrmsJobException;

GrmsJobInformation {
```



```
GrmsProjectIdentifier project;  
String userDn;  
GrmsJobStatusType status;  
Calendar submissionTime;  
Calendar finishTime;  
String errorDescription;  
GrmsTaskIdentifier[] tasksIdentifiers;  
int tasksCount;  
GrmsJobDescription jobDescription;  
}
```

Returns complex information about the job (project, owner, status, submission time, start time, finish time, error message, list of task, number of tasks, job description).

**GrmsJobInformation:**

- project – project, which the job belongs to,
- userDn – user Distinguish Name,
- status – status of the job,
- 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.

### **getTaskInformation.**

```
GrmsTaskInformation getTaskInformation( GrmsJobIdentifier, GrmsTaskIdentifier) \  
    throws GrmsGetTaskInformationException;
```

```
GrmsGetTaskInformationException extends GrmsTaskException;
```

```
GrmsTaskInformation {  
    GrmsTaskStatusType status;  
    Calendar submissionTime;  
    Calendar finishTime;  
    String requestStatus;  
    int reqNumStatus;  
    String errorDescription;  
    GrmsTaskHistory lastHistory;  
    int historyLength;  
}
```

Returns complex information about the task (status, submission time, start time, finish time, error message, list of task, number of tasks, latest history info, length of the history, job description).

**GrmsTaskInformation:**

- status – status of the task,
- submissionTime – submission time,
- finishTime – finish time or null if the time is unknown,
- requestStatus - status of GRMS request,
- reqNumStatus - numerical status of the request,



- `errorDescription` – message describing the cause of last error,
- `lastHistory` - information concerning the latest execution,
- `historyLength` - length of the history.

**getTaskHistory.**

```
GrmsTaskHistory[] getTaskHistory( GrmsJobIdentifier, GrmsTaskIdentifier) \
    throws GrmsGetTaskHistoryException;
```

```
GrmsGetTaskHistoryException extends GrmsTaskException;
```

```
GrmsTaskHistory {
    String hostName;
    Calendar startTime;
    Calendar localSubmissionTime;
    Calendar localStartTime;
    Calendar localFinishTime;
    GrmsTaskDescription taskDescription;
    GrmsTaskApplicationAccess applicationAccess;
}
```

Returns the information about the history of the task.

`GrmsTaskHistory`:

- `hostName` – name of the host, which the job is/was executed on,
- `startTime` – time when the job was submitted to the local resource,
- `localSubmissionTime` – time when the job was submitted on the local resource,
- `localStartTime` – time when the job was started on the local resource
- `localFinishTime` – time when the job was finished on the local resource,
- `taskDescription` - description of the task - part of job description concerning the task,
- `applicationAccess` - location of the service that can be used to checkpoint the application and the process identifier of application.

**findResources.**

```
GrmsResource[] findResources( GrmsTaskIdentifier, GrmsJobDescription) \
    throws GrmsFindResourcesException;
```

```
GrmsFindResourcesException extends GrmsTaskException;
```

Returns a list of resources (in the form of resource manager contact strings), where a specified task can be executed.

**testJobDescription.**

```
void testJobDescription( GrmsJobDescription) \
    throws GrmsTestJobDescriptionException;
```

```
GrmsTestJobDescriptionException extends GrmsException;
```

Tests the job description for syntax errors.

**registerJobNotification.**

```
GrmsNotificationIdentifier registerJobNotificaton( GrmsJobIdentifier, GrmsJobNotificationRequest) \
    throws GrmsRegisterJobNotifictionRequest;
```

```
GrmsRegisterJobNotificationException extends GrmsJobException;
```

```
GrmsJobNotificationRequest {
    GrmsNotificationProtocolType protocol;
    String destination;
    boolean appendMode;
    String format;
}
```

Registers the client for GRMS notifications concerning changes of status of the job and returns the identifier of the registered notification.

GrmsJobNotificationRequest:

- protocol - notification protocol: SOAP or GASS
- destination - notification end point,
- appendMode - in case of GASS notification determines if next notification should be appended at the end of file or should overwrite its content,
- format - determines format of the message (allows to register different patterns for notifications). For usage see appropriate section in "Grms functionality" chapter.

#### **unregisterJobNotification.**

```
void unregisterJobNotification( GrmsJobIdentifier, GrmsNotificationIdentifier) \
    throws GrmsUnregisterJobNotificationException;
```

```
GrmsUnregisterJobNotificationException extends GrmsJobNotification;
```

Unregisters the notification with given identifier.

#### **getJobNotificationsList.**

```
NotificationIdentifier[] getJobNotificationsList( GrmsJobIdentifier) \
    throws GrmsGetJobNotificationsListException;
```

```
GrmsGetJobNotificationsList extends GrmsJobException;
```

Returns list of notifications registered for the given job.

#### **getJobNotificationInformation.**

```
GrmsJobNotificationRequest getJobNotificationInformation( GrmsJobIdentifier, GrmsNotificationIdentifier) \
    throws GrmsGetJobNotificationInformationException;
```

```
GrmsGetJobNotificationException extends GrmsJobNotification;
```

Returns information about the registered notification.

#### **registerTaskNotification.**

```
GrmsNotificationIdentifier registerTaskNotification( GrmsJobIdentifier, GrmsTaskIdentifier, GrmsJobNotificationRequest) \
    throws GrmsRegisterTaskNotificationException;
```

```
GrmsRegisterTaskNotificationException extends GrmsTaskException;
```

```
GrmsTaskNotificationRequest extends GrmsJobNotificationRequest{
    GrmsNotificationEventType eventType;
}
```

Registers the client for GRMS notifications for changes concerning the task. For more details please see description of registerJobNotification method.

GrmsTaskNotificationRequest:

- eventType - type of the event: REQUEST or STATUS

**unregisterTaskNotification.**

```
void unregisterTaskNotificaton( GrmsJobIdentifier, GrmsTaskIdentifier, GrmsNotificationIdentifier) \
    throws GrmsUnregisterTasknotificationException;
```

```
GrmsUnregisterTasknotification extends GrmsTaskException;
```

Unregisters the notification with given identifier.

**getTaskNotificationsList.**

```
NotificationIdentifier[] getTaskNotificationsList( GrmsJobIdentifier, GrmsTaskIdentifier) \
    throws GrmsGetTaskNotificationsList;
```

```
GrmsGetTaskNotificationsList extends GrmsTaskException;
```

Returns list of notifications registered for the given task.

**getTaskNotificationInformation.**

```
GrmsTaskNotificationRequest getTaskNotificatonInformation( GrmsJobIdentifier, GrmsTaskIdentifier, GrmsNot
    throws GrmsGetTaskNotificationInformationexception;
```

```
GrmsGetTaskNotificationException extends GrmsTaskException;
```

Returns information about the registered notification.

**addTaskOutputFileDirs.**

```
GrmsFilesDirsResponse addTaskOutputFileDirs(GrmsJobIdentifier, GrmsTaskIdentifier, GrmsFileDir[]) \
    throws GrmsAddOutputFileDirsException;
```

```
GrmsAddAoutputFileDirsException extends GrmsTaskException;
```

```
GrmsFileDir {
    String name;
    String path;
    GrmsUrlType urltype;
    GrmsPathType pathtype;
}
```

```
GrmsFilesDirsResponse {
    int errorsCount;
    GrmsResponse[] response;
}
```

```
GrmsResponse {
    int errorCode;
    String errorMessage;
}
```

Registers a list of output files/directories, returns a structure containing array of corresponding diagnostic information and an aggregated number of errors.

GrmsFileDir:

- name – name of the file/directory
- path - location of the file/directory
- urltype – type of the location: PHYSICAL or LOGICAL
- pathtype – type of the entry: FILE or DIRECTORY

GrmsFilesDirsResponse:

- errorCount – number of errors



- response - list of diagnostic information describing status of operation for a single file or directory

GrmsResponse:

- errorCode – numerical status of operation
- errorMessage - error message describing the status (for example cause on failure).

### **getTaskOutputFileDirs.**

```
GrmsFileDir[] getTaskOutputFileDirs( GrmsJobIdentifier, GrmsTaskIdentifier) \
    throws GrmsGetTaskOutputFileDirsException;
```

```
GrmsGetTaskOutputFileDirsException extends GrmsTaskException;
```

Returns an array of output files/directories. See also addTaskOutputFileDirs.

### **deleteTaskOutputFileDirs.**

```
GrmsFilesDirsResponse deleteTaskOutputFileDirs( GrmsJobIdentifier, GrmsTaskIdentifier, GrmsFileDir[]) \
    throws GrmsDeleteTaskOutputFileDirsException;
```

```
GrmsDeleteTaskOutputFileDirsException extends GrmsTaskException;
```

Unregisters a list of output files/directories, returns a structure containing array of corresponding diagnostic information and an aggregated number of errors. See also addTaskOutputFileDirs.

### **addTaskCheckpointFileDirs.**

```
GrmsFilesDirsResponse addTaskCheckpointFileDirs(GrmsJobIdentifier, GrmsTaskIdentifier, GrmsFileDir[]) \
    throws GrmsAddTaskCheckpointFileDirsException;
```

```
GrmsAddTaskCheckpointFileDirs extends GrmsTaskException;
```

Registers a list of checkpoint files/directories, returns an array of corresponding diagnostic information and an aggregated number of errors. See also addTaskOutputFileDirs.

### **getCheckpointFileDirs.**

```
GrmsFileDir[] getCheckpointFileDirs( GrmsJobIdentifier, GrmsTaskIdentifier) \
    throws GrmsGetCheckpointFileDirsException;
```

```
GrmsGetCheckpointFileDirs extends GrmsTaskException;
```

Returns a list of checkpoint files/directories. See also addTaskOutputFileDirs.

### **deleteCheckpointFileDirs.**

```
GrmsFilesDirsResponse deleteTaskCheckpointFileDirs(GrmsJobIdentifier, GrmsTaskIdentifier, GrmsFileDir[]) \
    throws GrmsDeleteTaskCheckpointFileDirsException;
```

```
GrmsDeleteTaskCheckpointFileDirsException extends GrmsTaskException;
```

Unregisters a list of checkpoint files/directories, returns an array of corresponding diagnostic information and an aggregated number of errors. See also addTaskOutputFileDirs.

### **getServiceDescription.**

```
String getServiceDescription( GrmsDescriptionType) \
    throws GrmsGetServiceDescriptionException;
```

```
GrmsGetServiceDescriptionException extends GrmsException;
```

Returns a description of the service in the form of short or full format. GrmsDescriptionFile is an enumerated type and has two values: SHORT and FULL.

### **extendTaskExecutionTime.**



```
void extendTaskExecutionTime( GrmsJobIdentifier, GrmsTaskIdentifier, Duration) \
    throws GrmsExtendTaskExecutionTimeException;
```

```
GrmsExtendTaskExecutionTimeException extends GrmsTaskException;
```

Allows to extends task execution time specified in job description.

## GRMS Job Description (GJD)

As it was discussed in the previous section all job 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 as well as all job resource requirements. The following parameters are available in the GRMS Job Description:

- job executable:
  - location of the file,
  - arguments,
  - file arguments (files that have to be present 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 job 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,
  - there are some other parameters passed directly to GRAM (maxtime, maxwalltime, maxcputime),
  - network parameters (bandwidth, latency and capacity).
- constraints:
  - resource constraints
  - network constraints



- job execution time:
  - job execution time
  - time slot (e.g. from 10.00 till 16.00)
  - time period (e.g. till 31st March except Sundays)
- workflow:
  - parents of a task

Locations of files can be specified as gridFTP/GASS urls as well as logical ones using Replica System paths (for more information about data management services please visit [www.gridlab.org](http://www.gridlab.org)).

## Brief Specification of GJD

<grmsJob>

GRMS Job Description starts with the <grmsjob> tag, which contains the attribute "appid" that is an identifier (assigned by the user) of the application. The tag <grmsjob> has to contain <task> tags.

<task>

this tag is used for describing a single task, which is generally an 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. The <task> tag has to contain the <executable> tag and optionally can have <resource> tags.

<resource>

this tag is used to describe resource requirements for the job execution. Resource description can (optionally) 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,
<localrmname>	local resource management system on the host; acceptable values are: "fork" (default value), "lsf", "pbs", "sge" or "condor",

<memory>	minimal amount of the memory in MB,
<cpucount>	minimal number of processors,
<cpuspeed>	minimal speed of cpu(s),
<maxtime>	he maximum walltime or cputime for a single execution of the executable (in minutes),
<maxwalltime>	he maximum walltime for a single execution of the executable (in minutes),
<maxcputime>	he maximum cputime for a single execution of the executable (in minutes).
<bandwidth>	is a measure for the amount of network bandwidth that is 'unused' or in other words available between two hosts (in MBs).
<latency>	denotes the minimal time required to send a message on another host (in seconds).
<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).
<applications>	<p>is a list of required applications that have to be installed on a destination host. Every single &lt;application&gt; tag contains two optional attributes: "version" and "instanceCount".</p> <p>The "version" attribute denotes a required version of an application.</p> <p>The "instanceCount" attribute defines a required number of application instances that must be started in order to execute the main job.</p>
<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 number of required 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.

<executable>

describes the executable. It contains "type", "count" and "checkpointable" attributes and must have either <file> or <application> tag. Optionally, it can have the following single tags: <arguments>, <environment>, <checkpoint>, <stdin>, <stdout>, and <stderr>.

The "count" attribute denotes a number of executions of the executable.

The "type" attribute specifies a way 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 - start count processes or threads,
- mpi - use the appropriate method to start the job compiled with a vendor-provided MPI library. The job is started with count nodes.

The "checkpointable" attribute determines whether a job can be checkpointed or not.

<file>

this tag can be used to describe a file, which can be an executable as well as an argument file. It has to have the following attributes:

- "type" - that can have one of the values: "in" (for input file), "out" (for output file), "inout" (for input/output file). In the case of <executable>, the value of "type" attribute is ignored,
- "name", which denotes a name of file that, that the file should have after copying from <url> location,

and has to contain <url> or <collectionfile> tag.

<application>

defines an application that is installed on a destination host. It may contain the "version" attribute.

<url>

denotes url-location of file (gass http, grid ftp),

<collectionfile>

is used to describe a logical file from the replica management system (in particular, the Replica Catalogue, see [www.gridlab.org](http://www.gridlab.org) for more details). It contains the "name" attribute and the value. This schema was designed for the first version of the Replica Catalogue, but it is also possible to use it with the second one. The logical path is the concatenation of the "name" attribute and the value of tag.

<value>

describes command line arguments of the executable and/or files that have to be staged into a working directory before the execution or staged out after the job finished. It can contain also one or more <value> or <file> or <directory> tags.

<directory>

can be used to describe a directory, which is needed for the execution of the job or has to be transferred after the job is done. It contains "type" and "name" attributes (see <file> tag description) and contains <url> tag (directory url) or <collection> tag.

<collection>

it has similar meaning to the <collectionfile> tag. The difference is that it does not contain the value because collection is treated as a logical directory.

<stdin>

denotes standard input for the executable and contains the <url> tag (the same meaning as for <file> tag), or the <collectionfile> tag,

<stdout>

denotes standard output for the executable and contains the <url> tag (the same meaning as for <file> tag), or <collectionfile> tag,

<stderr>

denotes standard error for the executable and contains the <url> tag (the same meaning as for <file> tag), or <collectionfile> tag,

<environment>

can be used to describe environment variables for the job execution. It contains the attribute "name", which denotes a name of variable and contains its value,

<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> tag or <directory> tag.

<executionTime>

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

<execDuration>

specifies execution time of a job in min-

utes (i.e. it defines length of the period when a resource reservation is needed for a job).

<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> attribute and either the <slotEnd> or <slotDuration> attributes.

<slotStart>

specifies start time of the slot (as time of a day). A job must be started after this time.

<slotEnd>

specifies end time of the slot (as time of a day). A job must be started before this time.

<slotDuration>

specifies duration time of a slot. A job must be started before slot duration time ends.

<timePeriod>

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

<periodEnd>

specifies the end of a period (e.g. 12th February 2005 16:00PM).

<periodStart>

specifies start time of a period during which a job must be started (e.g. 31st January 10.00AM).

<periodDuration>

specifies duration of the time period (e.g. one week). If <periodStart> is not specified a default value of period start time is current time.

<including>

restricts a period when a job can be executed to certain days of a week (using the <weekDay> tag) and/or dates (using the <dateDay> tag), e.g. execute a job only on Fridays.

<excluding>

excludes certain days of a week (using the <weekDay> tag) and/or dates (using

the <dateDay> tag) from a period when a job can be executed, e.g. do not execute jobs on Sunday.

<constraints>

allows defining advanced constraints. It must contain at least one either <resConstraint> or <netConstraints> element, which extends the <baseConstraintType> type. Both <resConstraint> and <netConstraints> have the "parameter" attribute that specifies parameter on which constraint is imposed (e.g. CPUSpeed).

<resConstraints>

defines constraints concerning computational resources (nodes, clusters etc.)

<netConstraints>

defines constraints concerning network resources (e.g. bandwidth) between destination host and a host specified in the <endpoint> element

<baseConstraintsType>

includes a definition of soft and hard constraints

<hardConstraints>

are constraints that must be met (e.g. number of CPUs in range <4,32>). Either the exact value (<value> tag) or minimal and/or maximal values (<min>, <max> tags) can be defined. For each of these values the "indiffThreshold" attribute can be specified. The indifference threshold is a maximal difference between required and actual value of parameter such that a resource meets a constraint (e.g. for constraint Memory > 50MB and indiffThreshold=5MB, resource having Memory=45MB meets this constraint and resource having Memory=44MB does not).

<softConstraints>

expresses user's preferences (e.g. to find a machine characterized by 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 two mandatory attributes "preferenceType" and "optimizationType", and optional "indiffThreshold". It must contain the <importance> element that defines importance of this soft constraint.

The "preferenceType" attribute determines a method of expressing user's preferences. Two methods are sup-

ported:

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

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 the better resource
- **COST**- the less value of parameter the better resource

<workflow>

defines workflow of tasks. It must contains a list of parents. It can also have the optional "parentStates" attribute ("AND" is a default value) 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).

<parent>

specifies parents of this task. It can have two optional attributes: "triggerState" and "runSameHost". The former , defines a parent state after which this task can be started, and the latter specifies whether this task has to be run at the same host as its parent.

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,



---

CACTUS_LOCATION	location of the Cactus on the remote host,
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

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

2. 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="appid">
  <task taskid="taskid">
    <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`.

3. 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="appid">
  <task taskid="taskid">
    <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>
```

4. 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://ragel.man.poznan.pl/~examples/date`. (“~” denotes your working directory). You can also specify the whole gsiftp path to your executable (`gsiftp://ragel.man.poznan.pl//home/user1/examples/date`). Note that there are two slashes after the address of machine.

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

5. If the execution of task should be preceded by getting an executable file from the best location registered in the Replica Location Catalogue, the <logicalId> tag should be used. Let’s assume that the logical path to the file is “/home/piontek/examples/date”. Using the optional "user" attribute it is possible to specify location which is relative to the user's logical home directory.

```
<grmsJob appid="appid">
  <task taskid="taskid">
    <executable type="single" count="1">
      <execfile name="exec-file">
        <logicalId>/home/piontek/examples/date</logicalId>
      </execfile>
    </executable>
  </task>
</grmsJob>
```

```
<grmsJob appid="appid">
  <task taskid="taskid">
    <executable type="single" count="1">
      <execfile name="exec-file">
        <logicalId user="/C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek">examples/date</logicalId>
      </execfile>
    </executable>
  </task>
</grmsJob>
```

6. 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 follows:

```
<grmsJob appid="appid">
  <task taskid="taskid">
    <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>
```

7. 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="appid">
  <task taskid="taskid">
    <executable type="single" count="1">
      <execfile name="exec-file">
        <url>file:///bin/cat</url>
      </execfile>
      <arguments>
        <value>file.log</value>
      </arguments>
    </executable>
  </task>
</grmsJob>
```

```

    <file name="file.log" type="in">
      <url>gsiftp://ragel.man.poznan.pl/~examples/file.log</url>
    </file>
  </arguments>
</executable>
</task>
</grmsJob>

```

8. 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://ragel.man.poznan.pl/~examples/report.tar”.

```

<grmsJob appid="appid">
  <task taskid="taskid">
    <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://ragel.man.poznan.pl/~examples/report</url>
        </file>
        <file name="report.tar" type="out">
          <url>gsiftp://ragel.man.poznan.pl/~examples/report.tar</url>
        </file>
      </arguments>
    </executable>
  </task>
</grmsJob>

```

9. 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** and then read stdin from the file “stdin\_file”.

```

<grmsJob appid="appid">
  <task taskid="taskid">
    <executable type="single" count="1">
      <execfile name="exec-file">
        <url>file:///bin/cat</url>
      </execfile>
      <stdin>
        <url>gsiftp://ragel.man.poznan.pl/~examples/stdin_file</url>
      </stdin>
    </executable>
  </task>
</grmsJob>

```

10. 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 file defined by the URL: “gsiftp://ragel.man.poznan.pl/~examples/grep\_output”.

```

<grmsJob appid="appid">
  <task taskid="taskid">
    <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://ragel.man.poznan.pl/~examples/grep_input</url>
        </file>
      </arguments>
      <stdout>
        <url>gsiftp://ragel.man.poznan.pl/~examples/grep_output</url>
      </stdout>
    </executable>
  </task>

```

```
</grmsJob>
```

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

12. If a task must be migrated (for instance, because of its poor performance), GRMS Job Description has to contain all checkpoint information that defines the current state of job execution. Let's assume that we want to start the application called "gattest-static", which needs for its execution the following files: paramfile, adaptors, testsoap.so on the "host ragel.man.poznan.pl". The application is started with "paramfile", "200" arguments and during the execution the application stores some debug information in the "log" file. The jobDescription starting the application is presented below:

```
<grmsjob appid="migration">
  <task taskid="taskid">
    <resource>
      <hostname>ragel.man.poznan.pl</hostname>
    </resource>
    <executable type="single" count="1" checkpointable="true">
      <execfile name="gatapp">
        <url>gsiftp://ragel.man.poznan.pl/~gatapp/gattest-static</url>
      </execfile>
      <arguments>
        <value>paramfile</value>
        <value>200</value>
        <file name="paramfile" type="in">
          <url>gsiftp://ragel.man.poznan.pl/~gatapp/paramfile</url>
        </file>
        <file name="adaptors" type="in">
          <url>gsiftp://ragel.man.poznan.pl/~gatapp/adaptors</url>
        </file>
        <file name="testsoap.so" type="in">
          <url>gsiftp://ragel.man.poznan.pl/~gatapp/testsoap.so</url>
        </file>
      </arguments>
    </executable>
  </task>
</grmsJob>
```

Let's assume that because of the application poor performance we want to checkpoint and migrate it to the host "rage2.man.poznan.pl". The checkpoint operation stores all information needed to restart the application in the "checkpoint.random" file. In this case, additionally to information needed to start application, we have to tell GRMS about two files: "log" and "checkpoint.random". If we assume that both files are placed in the application work directory we don't have to specify their `<url>`s. When the application is finished, the "log" file should be copied to a location defined by the user.

```
<grmsJob appid="migration">
  <task>
    <resource>
      <hostname>rage2.man.poznan.pl</hostname>
    </resource>
    <executable type="single" count="1">
```

```

<execfile name="gatapp">
  <url>gsiftp://ragel.man.poznan.pl/~gatapp/gatatest-static</url>
</execfile>
<arguments>
  <value>paramfile</value>
  <file name="paramfile" type="in">
    <url>gsiftp://ragel.man.poznan.pl/~gatapp/paramfile</url>
  </file>
  <file name="adaptors" type="in">
    <url>gsiftp://ragel.man.poznan.pl/~gatapp/adaptors</url>
  </file>
  <file name="testsoap.so" type="in">
    <url>gsiftp://ragel.man.poznan.pl/~gatapp/testsoap.so</url>
  </file>
  <file name="log" type="out">
    <url>gsiftp://ragel.man.poznan.pl/~examples/migration_results</url>
  </file>
</arguments>
<checkpoint>
  <file name="checkpoint.random" type="in">
    <url></url>
  </file>
  <file name="log" type="in">
    <url></url>
  </file>
</checkpoint>
</executable>
</task>
</grmsJob>

```

13. This example presents the usage of GRMS environment variables.

```

<grmsJob appid="Simulation">
  <task taskid="taskid">
    <resource>
      <hostname>skirit.ics.muni.cz</hostname>
    </resource>
    <executable type="single" count="1">
      <execfile name="cactus_go.sh">
        <url>file:///${CACTUS_LOCATION}/cactus_go.sh</url>
      </execfile>
      <arguments>
        <value>./bbh-64.par</value>
        <file name="bbh-64.par" type="in">
          <url>file:///${HOME}/demo/par/bbh-64.par</url>
        </file>
      </arguments>
      <checkpoint>
        <directory name="${JOB_ID}.checkpoint" type="in">
          <collection name="/home/glab040/${JOB_ID}.checkpoint"/>
        </directory>
      </checkpoint>
    </executable>
  </task>
</grmsJob>

```

14. This example presents the usage of "persistent" and "extension" attributes. Let us assume that we want to pack (tar) the output of **/bin/ps -ef** command and copy the obtained archive to a 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: the first task will execute the aforementioned command and catch its output to a specified file (the task will be specified as a persistent one, so grms will not remove its directory when it is finished), then the next task will pack the file and copy it to a remote location (the second task will be specified as an extension of the first one, so it 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="ps_tar">
  <task taskid="taskid_1" persistent="true">
    <resource>
      <hostname>ragel.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="taskid_2">
  <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://rage1.man.poznan.pl/~ps.out.tgz</url>
      </file>
    </arguments>
  </executable>
</workflow>
  <parent triggerState="FINISHED">taskid_1</parent>
</workflow>
</task>
</grmsJob>

```

15. 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="ps_tar">
  <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_output">
    <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="rage3_tar">
    <resource>
      <hostname>rage3.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>ps.outs.tgz</value>
  <value>rage?_ps.out</value>
  <file name="ps.outs.tgz" type="out">
    <url>gsiftp://rage4.man.poznan.pl/~ps.out.tgz</url>
  </file>
</arguments>
</executable>
<workflow parentStates="AND">
  <parent triggerState="FINISHED">rage1_ps</parent>
  <parent triggerState="FINISHED">rage2_ps</parent>
</workflow>
</task>
</grmsJob>
```

## GRMS java client

### Requirements.

- Java - JDK 1.4.x
- ant 1.6.x

### Important

There is a known bug connected with OGSA stuff (used in GRMS to provide Transport Level Security for SOAP communication) and new versions of Java starting from SUN JDK 1.4.2\_05. It is caused by the fact that these new versions of JRE have bundled a newer version of Xalan, which made an originally public variable private, and the Apache XML Security library used by OGSA is incorrectly accessing it. This cause that client code breaks with following exception:

```
java.lang.IllegalAccessException: org.apache.xml.security.Init tried to access \
    field org/apache/xpath/compiler/FunctionTable.m_functions from class
    at org.apache.xml.security.Init.init(Unknown Source)
    at org.globus.ogsa.impl.security.authentication.wssec.WSSecurityEngine.\
        <clinit>(WSSecurityEngine.java:55)
    at org.globus.ogsa.impl.security.authentication.wssec.\
        WSSecurityClientHandler.handleResponse(WSSecurityClientHandler.java:51)
    at org.apache.axis.handlers.HandlerChainImpl.\
        handleResponse(HandlerChainImpl.java:153)
```

A workarounds are either to use SUN JDK 1.4.2\_04, IBM JDK 1.4.1 or put xalan.jar from  
<grms\_location>/jars/ogsa/xalan.jar into  
\$JAVA\_HOME/jre/lib/endorsed/xalan.jar

Form more details please see:

- [http://issues.apache.org/bugzilla/show\\_bug.cgi?id=30764](http://issues.apache.org/bugzilla/show_bug.cgi?id=30764)
- [http://www-unix.globus.org/mail\\_archive/discuss/2004/09/msg00017.html](http://www-unix.globus.org/mail_archive/discuss/2004/09/msg00017.html)



---

## Installation and configuration of GRMS client

1. Unpack GRMS v.2.0 release

2. Compile the client

```
$> ant -f build.interface.xml jar
```

3. Edit `<grms_location>/bin/ws_client.sh` script

Please modify following lines if it is necessary:

```
GRMS_URL="http://druid-bis.man.poznan.pl:8443/axis/services/grms"
GRMS_DN="/C=PL/O=GRID/O=PSNC/CN=grms_devel/ragel.man.poznan.pl"
GRMS_TIMEOUT=5
GRMS_DELEG_TYPE=FULL
```

- 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.
- 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.

4. Create or modify `cog.properties` file in `~/globus` directory, change values of properties according to your configuration

```
#Java CoG Kit Configuration File
#Wed Jul 17 09:25:01 CEST 2002
usercert=../../usercert.pem
userkey=../../userkey.pem
proxy=/tmp/x509up_u501
cacert=/etc/grid-security/certificates/
```

For details please visit: <http://www-unix.globus.org/cog/distribution/1.1/FAQ.TXT>

## Usage of GRMS commandline client

### Executing GRMS Client

1. run `<globus_location>/bin/grid-proxy-init` command to create user's proxy

2. run GRMS client typing in `<grms_location>/bin` directory.  
`./ws_client {operation} [argument...]`

### Usage of GRMS

```
./ws_client.sh submit_job <jobDescriptionFile>
./ws_client.sh migrate_job <jobID> [<jobDescriptionFile>]
./ws_client.sh suspend_job <jobId> [<jobDescriptionFile>]
./ws_client.sh resume_job <jobId> [<jobDescriptionFile>]
./ws_client.sh cancel_job <jobId>
```



```
./ws_client.sh submit_task <taskId> <jobDescriptionFile>
./ws_client.sh migrate_task <jobId> <taskId> [<jobDescriptionFile>]
./ws_client.sh suspend_task <jobId> <taskId> [<jobDescriptionFile>]
./ws_client.sh resume_task <jobId> <taskId> [<jobDescriptionFile>]
./ws_client.sh canceltask <jobId> <taskId>

./ws_client.sh list_all SUBMITTED | ACTIVE | FINISHED | SUSPENDED | FAILED | CANCELED

./ws_client.sh list_jobs [SUBMITTED | ACTIVE | FINISHED | SUSPENDED | FAILED | CANCELED]

./ws_client.sh list_tasks <jobId> [QUEUED | PREPROCESSING | PENDING | RUNNING | STOPPED |
    POSTPROCESSING | FINISHED | SUSPENDED | FAILED | CANCELED]

./ws_client.sh list_project [SUBMITTED | ACTIVE | FINISHED | SUSPENDED | FAILED | CANCELED]

./ws_client.sh register_access <jobId> <taskId> <service_location> <pid>
./ws_client.sh unregister_access <jobid> <taskId>
./ws_client.sh get_access <jobId> <taskId>

./ws_client.sh job_info <jobId>
./ws_client.sh task_info <jobId> <taskId>
./ws_client.sh task_history <jobId> <taskId>

./ws_client.sh resources <taskId> <jobDescriptionFile>

./ws_client.sh test <jobDescriptionFile>

./ws_client.sh add_job_notif <jobid> SOAP|GASS <destination> [true|false [<format>]]
./ws_client.sh del_job_notif <jobid> <notificationId>
./ws_client.sh list_job_notif <jobid>
./ws_client.sh info_job_notif <jobid> <notificationId>

./ws_client.sh add_task_notif <jobid> <taskId> STATUS|REQUEST SOAP|GASS <destination> [true|false [<forma
./ws_client.sh del_task_notif <jobid> <taskId> <notificationId>
./ws_client.sh list_task_notif <jobid> <taskId>
./ws_client.sh info_task_notif <jobid> <taskId> <notificationId>

./ws_client.sh add_output <jobId> <taskId> <name> <path> PHYSICAL|LOGICAL FILE|DIRECTORY
./ws_client.sh get_output <jobId> <taskId>
./ws_client.sh del_output <jobId> <taskId> <name> <path> PHYSICAL|LOGICAL FILE|DIRECTORY
./ws_client.sh add_checkpoint <jobId> <taskId> <name> <path> PHYSICAL|LOGICAL FILE|DIRECTORY
./ws_client.sh get_checkpoint <jobId> <taskId>
./ws_client.sh del_checkpoint <jobId> <taskId> <name> <path> PHYSICAL|LOGICAL FILE|DIRECTORY

./ws_client.sh extend_execution <jobId> <taskId> <duration>

./ws_client.sh description SHORT|FULL
```

## Important

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

## Examples of execution

### General issues

- 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 bin]$ ./ws_client.sh suspend_job 1085556664951_appid_1620
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: http://druid-bis.man.poznan.pl:8443/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 bin]$ ./ws_client.sh cancel_job 1085556664951_appid_1620
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: httpg://druid-bis.man.poznan.pl:8443/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 bin]$ ./ws_client.sh job_info non-existing-job-ID
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: httpg://druid-bis.man.poznan.pl:8443/axis/services/grms
- Getting job info failed!
- GRMS response was:
- errorCode: 114
- errorMessage: No such job in Job Repository
```

```
[piontek@druid bin]$ ./ws_client.sh task_info 1085556664951_appid_1620 non-existing-task-ID
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: httpg://druid-bis.man.poznan.pl:8443/axis/services/grms
- Getting job info failed!
- GRMS response was:
- errorCode: 117
- errorMessage: No such task in Task Repository
```

- 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 bin]$ ./ws_client.sh task_info 1083844628098_appid_2377
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: httpg://druid-bis.man.poznan.pl:8443/axis/services/grms
- Getting job info failed!
- GRMS response was:
- errorCode: 154
- errorMessage: User not authorized to perform operation - internal authorization
```

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

```
[piontek@druid bin]$ ./ws_client.sh submit_job ../examples/error.xml
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: httpg://druid-bis.man.poznan.pl:8443/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

```
./ws_client.sh submit_job {jobDescription}
```

Corresponds to the "submitJob" method. If the GRMS Job Description does not contain any syntax error, the identifier of the submitted job is displayed.

```
[piontek@druid bin]$ ./ws_client.sh submit_job ../examples/example2.xml
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: httpg://druid-bis.man.poznan.pl:8443/axis/services/grms
- Job submitted successfully, jobId=1085556664951_appid_1620
```

## migrate\_job

```
./ws_client.sh migrate_job {jobId} [jobDescription]
```

Corresponds to the "migrateJob" method. If the job can be migrated by the user (user is authorized to perform the migration, job is in proper state and belongs to the user) the migration process is started.

```
[piontek@druid bin]$ ./ws_client.sh migrate_job 1084456104272_appid_0728 ../examples/example2.xml
```



- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: <http://druid-bis.man.poznan.pl:8443/axis/services/grms>
- Migration in progress. Invoke `./ws_client.sh job_info` to check the status of the operation.

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

```
[piontek@druid bin]$ ./ws_client.sh migrate_job 1085141911671_appid_3916 ../examples/example2.xml
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: http://druid-bis.man.poznan.pl:8443/axis/services/grms
- Job migration failed!
- GRMS response was:
- errorCode: 142
- errorMessage: Job in current state can not be migrated
```

## suspend\_job

```
./ws_client.sh suspend_job {jobId} [jobDescription]
```

Corresponds to the "suspendJob" method. If the job can be suspended by the user (user is authorized to perform the migration, job is in proper state and belongs to the user) the process of suspending starts.

```
[piontek@druid bin]$ ./ws_client.sh suspend_job 1085141911671_appid_3916
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: http://druid-bis.man.poznan.pl:8443/axis/services/grms
- Suspending in progress. Invoke ./ws_client.sh job_info to check the status of the operation.
```

## resume\_job

```
./ws_client.sh resume_job {jobId} [jobDescription]
```

Corresponds to the "resumeJob" method. This functionality resumes task execution.

```
[piontek@druid bin]$ ./ws_client.sh resume 1085141911671_appid_3916 ../examples/resume.xml
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: http://druid-bis.man.poznan.pl:8443/axis/services/grms
- Resuming in progress. Invoke ./ws_client.sh job_info to check the status of the operation.
```

## submit\_task

```
./ws_client.sh submit_job {taskId} {jobDescription}
```

Corresponds to the "submitTask" method. If the GRMS Job Description does not contain any syntax error and the job description contains task with given taskId, the identifier of the submitted job is displayed.

```
[piontek@druid bin]$ ./ws_client.sh submit_job taskid_1 ../examples/example2.xml
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: http://druid-bis.man.poznan.pl:8443/axis/services/grms
- Task submitted successfully, jobId=1085556664951_appid_1620
```

## migrate\_task

```
./ws_client.sh migrate_task {jobId} {taskId} [jobDescription]
```

Corresponds to the "migrateTask" method. If the task can be migrated by the user (user is authorized to perform the migration, task is in proper state and belongs to the user) the migration process is started.

```
[piontek@druid bin]$ ./ws_client.sh migrate_task 1084456104272_appid_0728 taskid_1 ../examples/example2.x
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: http://druid-bis.man.poznan.pl:8443/axis/services/grms
- Migration in progress. Invoke ./ws_client.sh 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 migration it, the proper error message is displayed.

```
[piontek@druid bin]$ ./ws_client.sh migrate_task 1085141911671_appid_3916 taskid_1 ../examples/example2.x
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: http://druid-bis.man.poznan.pl:8443/axis/services/grms
- Job migration failed!
```



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

## suspend\_task

```
./ws_client.sh suspend_task {jobId} {taskId} [jobDescription]
```

Corresponds to the "suspendTask" method. If the task can be suspended by the user (user is authorized to perform the operation, task is in proper state and belongs to the user) the process of suspending starts.

```
[piontek@druid bin]$ ./ws_client.sh suspend_task 1085141911671_appid_3916 taskid_1
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: http://druid-bis.man.poznan.pl:8443/axis/services/grms
- Suspending in progress. Invoke "./ws_client.sh task_info" to check the status of the operation.
```

## resume\_task

```
./ws_client.sh resume_task {jobId} {taskId} [jobDescription]
```

Corresponds to the "resumeTask" method. This functionality resumes task execution.

```
[piontek@druid bin]$ ./ws_client.sh resume_task 1085141911671_appid_3916 taskid_1 ../examples/resume.xml
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: http://druid-bis.man.poznan.pl:8443/axis/services/grms
- Resuming in progress. Invoke "./ws_client.sh task_info" to check the status of the operation.
```

## cancel\_job

```
./ws_client.sh cancel_job {jobId}
```

Corresponds to the "cancelJob" method. If the job can be canceled by the user (user is authorized to perform the operation, job is in the proper state and belongs to the user), the job is stopped.

```
[piontek@druid bin]$ ./ws_client.sh cancel_job 1085141911671_appid_3916
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: http://druid-bis.man.poznan.pl:8443/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 bin]$ ./ws_client.sh cancel_job 1085141911671_appid_3916
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: http://ragel.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
```

## cancel\_task

```
./ws_client.sh cancel_task {jobId} {taskId}
```

Corresponds to the "cancelTask". If the task can be canceled by the user (user is authorized to perform the operation, task is in the proper state and belongs to the user), the task is stopped.

```
[piontek@druid bin]$ ./ws_client.sh cancel_task 1085141911671_appid_3916 taskid_1
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: http://druid-bis.man.poznan.pl:8443/axis/services/grms
- Task has been cancelled successfully
```

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

```
piontek@druid bin]$ ./ws_client.sh cancel_job 1085141911671_appid_3916 taskid_1
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: http://druid-bis.man.poznan.pl:8443/axis/services/grms
- Canceling of job failed!
- GRMS response was:
- errorCode: 314
```



- errorMessage: Task can not be canceled in current state

## list\_all

`./ws_client.sh list_all SUBMITTED | ACTIVE | FINISHED | SUSPENDED | FAILED | CANCELED`

Corresponds to the "getAlljobsList" method. If the argument is one of predefined values, the list of all jobs in given state is displayed.

```
[piontek@druid bin]$ ./ws_client.sh list_all canceled
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: httpg://druid-bis.man.poznan.pl:8443/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

`./ws_client.sh list_jobs [SUBMITTED | ACTIVE | FINISHED | SUSPENDED | FAILED | CANCELED]`

Corresponds to the "getJobsList" method. If the argument is one of predefined values the list of jobs in given state and belonging to the user is displayed.

```
[piontek@druid bin]$ ./ws_client.sh list_jobs canceled
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: httpg://druid-bis.man.poznan.pl:8443/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 bin]$ ./ws_client.sh list_jobs
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: httpg://druid-bis.man.poznan.pl:8443/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
```

## list\_tasks

`./ws_client.sh list_tasks {jobId} [QUEUED | PREPROCESSING | PENDING | RUNNING | STOPPED | POSTPROCESSING | FINISHED | SUSPENDED | FAILED | CANCELED]`

Corresponds to the "getTasksList" method. If the argument is one of predefined values the list of tasks in given state and belonging to the job.

```
[piontek@druid bin]$ ./ws_client.sh list_tasks failed
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: httpg://druid-bis.man.poznan.pl:8443/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 bin]$ ./ws_client.sh list_tasks
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: httpg://druid-bis.man.poznan.pl:8443/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

`./ws_client.sh list_project [SUBMITTED | ACTIVE | FINISHED | SUSPENDED | FAILED | CANCELED]`

Corresponds to the "getProjectJobsList" method. If the argument is one of predefined values the list of jobs in given state and belonging to the project is displayed.

```
[piontek@druid bin]$ ./ws_client.sh list_project gridlab_project
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: http://druid-bis.man.poznan.pl:8443/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 project are listed.

```
[piontek@druid bin]$ ./ws_client.sh list_project
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: http://druid-bis.man.poznan.pl:8443/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
```

## register\_access

`./ws_client.sh register_access {jobId} {taskId} {service-location} {pid}`

Corresponds to the "registerTaskApplicationAccess" method.

```
[piontek@druid bin]$ ./ws_client.sh register_access 1084367864146_appid_1408 task_1 \
http://ragel.man.poznan.pl:4567 1234
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: http://druid-bis.man.poznan.pl:8443/axis/services/grms
- Task Application Access registered successfully
```

## Important

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

## get\_access

`./ws_client.sh get_access {jobId} {taskId}`

Corresponds to the "getTaskApplicationAccess" method.

```
[piontek@druid-bis bin]$ ./ws_client.sh get_access 1084367864146_appid_1408 task_1
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: http://druid-bis.man.poznan.pl:8443/axis/services/grms
- Task Application Access got successfully
- ServiceLocation = http://ragel.man.poznan.pl:4567
- PID = 1234
```

## unregister\_access

`./ws_client.sh unregister_access {jobId} {taskId}`

Corresponds to the "unregisterTaskApplicationAccess" method.

```
[piontek@druid-bis bin]$ ./ws_client.sh unregister_access 1084367864146_appid_1408 task_1
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: http://druid-bis.man.poznan.pl:8443/axis/services/grms
- Task Application Access unregistered successfully
```

## job\_info

```
./ws_client.sh job_info {jobId}
```

Corresponds to the "getJobInformation" method.

```
[piontek@druid-bis bin]$ ./ws_client.sh info 1107243045980_ps_tar_3879
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: http://druid-bis.man.poznan.pl:8443/axis/services/grms
- jobInfo[1107243045980_ps_tar_3879] is:
- project=Guide
- userDN=/C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- status=FINISHED
- submissionTime=Fri Mar 18 08:30:45 CET 2005
- finishTime=Fri Mar 18 08:33:02 CET 2005
- errorDescription=
- tasks=taskid_1 taskid_2
- tasksNumber=2
- jobDescription=
<grmsJob appId="ps_tar" project="Guide">
  <task taskid="taskid_1" persistent="true">
    <resource>
      <hostname>ragel.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="taskid_2">
    <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://ragel.man.poznan.pl/~ps.out.tgz</url>
        </file>
      </arguments>
    </executable>
  <workflow>
    <parent triggerState="FINISHED">taskid_1</parent>
  </workflow>
</task>
</grmsJob>
```

## task\_info

```
./ws_client.sh task_info {jobId} {taskId}
```

Corresponds to "getTaskInformation" method.

```
[piontek@druid-bis bin]$ ./ws_client.sh task_info 1107243045980_ps_tar_3879
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: http://druid-bis.man.poznan.pl:8443/axis/services/grms
- taskInfo[1107243045980_ps_tar_3879, taskid_1] is:
- submissionTime=Fri Mar 18 08:30:51 CET 2005
- finishTime=Fri Mar 18 08:32:20 CET 2005
- status=FINISHED
- requestStatus=TASK_DONE
- reqNumStatus=13
- errorDescription=
- hostName=ragel.man.poznan.pl
- startTime=Fri Mar 18 08:30:59 CET 2005
- localSubmissionTime=Fri Mar 18 08:31:03 CET 2005
- localStartTime=Fri Mar 18 08:31:11 CET 2005
- localFinishTime=Fri Mar 18 08:32:07 CET 2005
- taskDescription=
```



```
<task taskid="taskid_1" persistent="true">
  <resource>
    <hostname>ragel.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>
- applicationAccess.location=http://ragel.man.poznan.pl:40005
- applicationAccess.pid=2781
- historyLength=1
```

## task\_history

```
./ws_client.sh task_history {jobId} {taskId}
```

Corresponds to "getTaskHistory" method.

```
[piontek@druid-bis bin]$ ./ws_client.sh history 1107243045980_ps_tar_3879 taskid_1
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: http://druid-bis.man.poznan.pl:8443/axis/services/grms
- jobHistory[1107243045980_ps_tar_3879, taskid_1] is:
- JobHistory[0]
- hostName=ragel.man.poznan.pl
- startTime=Fri Mar 18 08:30:59 CET 2005
- localSubmissionTime=Fri Mar 18 08:31:03 CET 2005
- localStartTime=Fri Mar 18 08:31:11 CET 2005
- localFinishTime=Fri Mar 18 08:32:07 CET 2005
- taskDescription=
  <task taskid="taskid_1" persistent="true">
    <resource>
      <hostname>ragel.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>
- applicationAccess.location=http://ragel.man.poznan.pl:40005
- applicationAccess.pid=2781
```

## resources

```
./ws_client.sh resources {taskId} {jobDescription}
```

Corresponds to the "findResources" method.

```
[piontek@druid bin]$ ./ws_client.sh resources task_1 ../examples/example1.xml
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: http://druid-bis.man.poznan.pl:8443/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.bcv.lsu.edu:2119/jobmanager-fork:/O=Grid/O=GridLab/CN=helix.bcv.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

```
./ws_client.sh test {jobDescription}
```

Corresponds to the "testJobDescription" method.

```
[piontek@druid bin]$ ./ws_client.sh test ../examples/example2.xml
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: http://druid-bis.man.poznan.pl:8443/axis/services/grms
- Job description is correct
```

```
[piontek@druid bin]$ ./ws_client.sh test ../examples/error.xml
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: http://druid-bis.man.poznan.pl:8443/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

```
./ws_client.sh add_job_notif {jobId} SOAP|GASS [true|false [format]]
```

Corresponds to the "registerJobNotification" method.

```
./ws_client.sh add_job_notif 1107243045980_ps_extension_3879 soap \
http://ragel.man.poznan.pl:1222
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: http://druid-bis.man.poznan.pl:8443/axis/services/grms
- Notification listener was registered successfully.
notificationId = 1107243045980_ps_extension_3879:1107246112047
```

```
[piontek@druid-bis bin]$ ./ws_client.sh add_job_notif 1107243045980_ps_extension_3879 \
gass http://ragel.man.poznan.pl:1222 true \
"Request status of the job %s has changed to %s."
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: http://druid-bis.man.poznan.pl:8443/axis/services/grms
- Notification listener was registered successfully.
notificationId = 1107243045980_ps_extension_3879:1107246248349
```

## list\_job\_notif

```
./ws_client.sh list_job_notif {jobId}
```

Corresponds to the "getJobNotificationsList" method.

```
[piontek@druid-bis bin]$ ./ws_client.sh list_job_notif 1107243045980_ps_extension_3879
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: http://druid-bis.man.poznan.pl:8443/axis/services/grms
- List of notifications:
- 1107243045980_ps_extension_3879:1107246112047
- 1107243045980_ps_extension_3879:1107246248349
```

## info\_job\_notif

```
./ws_client.sh info_job_notif {jobId} {notificationId}
```

Corresponds to the "getJobNotificationInformation" method.

```
[piontek@druid-bis bin]$ ./ws_client.sh info_job_notif 1107243045980_ps_ext_3879 \
1107243045980_ps_ext_3879:1107246112047
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: http://druid-bis.man.poznan.pl:8443/axis/services/grms
- notificationInfo[1107243045980_ps_ext_3879, 1107243045980_ps_ext_3879:1107246112047] is:
- Protocol=SOAP
```



```
- Destination=http://ragel.man.poznan.pl:1222

[piontek@druid-bis bin]$ ./ws_client.sh info_job_notif 1107243045980_ps_ext_3879 \  
1107243045980_ps_ext_3879:1107246248349 \  
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek \  
- Service URL: http://druid-bis.man.poznan.pl:8443/axis/services/grms \  
- notificationInfo[1107243045980_ps_ext_3879, 1107243045980_ps_ext_3879:1107246248349] is: \  
- Protocol=GASS \  
- Destination=http://ragel.man.poznan.pl:1222 \  
- AppendMode=true \  
- Format=Request status of the job %s has changed to %s.
```

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

```
[piontek@druid-bis bin]$ ./ws_client.sh info_job_notif 1107243045980_ps_ext_3879 wrong_notif_id \  
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek \  
- Service URL: http://druid-bis.man.poznan.pl:8443/axis/services/grms \  
- Getting Notification info failed! \  
- GRMS response was: \  
- errorCode: 165 \  
- errorMessage: Failed to get notification - no such notification
```

## del\_job\_notif

```
./ws_client.sh del_job_notif {jobId} {notificationId}
```

Corresponds to the "unregisterJobNotification" method.

```
[piontek@druid-bis bin]$ ./ws_client.sh 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: http://druid-bis.man.poznan.pl:8443/axis/services/grms \  
- Notification listener was unregistered successfully.
```

## add\_task\_notif

```
./ws_client.sh add_task_notif {jobId} {taskId} STATUS|REQUEST SOAP|GASS [true|false for-  
[mat]]
```

Corresponds to the "registerTaskNotification" method.

```
[piontek@druid-bis bin]$ ./ws_client.sh add_task_notif 1107243045980_ps_extension_3879 taskid_1 status so-  
http://ragel.man.poznan.pl:1222 \  
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek \  
- Service URL: http://druid-bis.man.poznan.pl:8443/axis/services/grms \  
- Notification listener was registered successfully. \  
notificationId = 1107243045980_ps_extension_3879:taskid_1:1107246112047

[piontek@druid-bis bin]$ ./ws_client.sh add_task_notif 1107243045980_ps_extension_3879 taskid_1 request \  
gass http://ragel.man.poznan.pl:1222 true \  
"Request status of the job %s has changed to %s." \  
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek \  
- Service URL: http://druid-bis.man.poznan.pl:8443/axis/services/grms \  
- Notification listener was registered successfully. \  
notificationId = 1107243045980_ps_extension_3879:taskid_1:1107246248349
```

## info\_task\_notif

```
./ws_client.sh info_task_notif {jobId} {taskId} {notificationId}
```

Corresponds to the "getTasknotificationInformation" method.

```
[piontek@druid-bis bin]$ ./ws_client.sh info_task_notif 1107243045980_ps_ext_3879 taskid_1 \  
1107243045980_ps_ext_3879:taskid_1:1107246112047 \  
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek \  
- Service URL: http://druid-bis.man.poznan.pl:8443/axis/services/grms \  
- notificationInfo[1107243045980_ps_ext_3879, 1107243045980_ps_ext_3879:taskid_1:1107246112047] is: \  
- EventType=STATUS \  
- Protocol=SOAP \  
- Destination=http://ragel.man.poznan.pl:1222
```



```
[piontek@druid-bis bin]$ ./ws_client.sh info_task_notif 1107243045980_ps_ext_3879 taskid_1 \  
1107243045980_ps_ext_3879:taskid_1:1107246248349  
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek  
- Service URL: http://druid-bis.man.poznan.pl:8443/axis/services/grms  
- notificationInfo[1107243045980_ps_ext_3879, 1107243045980_ps_ext_3879:taskid_1:1107246248349] is:  
- EventType=REQUEST  
- Protocol=GASS  
- Destination=http://ragel.man.poznan.pl:1222  
- AppendMode=true  
- Format=Request status of the job %s has changed to %s.
```

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

```
[piontek@druid-bis bin]$ ./ws_client.sh info_task_notif 1107243045980_ps_ext_3879 taskid_1 wrong_notif_id  
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek  
- Service URL: http://druid-bis.man.poznan.pl:8443/axis/services/grms  
- Notification unregistration failed!  
- GRMS response was:  
- errorCode: 165  
- errorMessage: Failed to get notification - no such notification
```

## list\_task\_notif

```
./ws_client.sh list_task_notif {jobId} {taskId}
```

Corresponds to the "getTaskNotificationsList" method.

```
[piontek@druid-bis bin]$ ./ws_client.sh list_task_notif 1107243045980_ps_extension_3879 taskid_1  
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek  
- Service URL: http://druid-bis.man.poznan.pl:8443/axis/services/grms  
- List of notifications:  
- 1107243045980_ps_extension_3879:taskid_1:1107246112047  
- 1107243045980_ps_extension_3879:taskid_1:1107246248349
```

## del\_task\_notif

```
./ws_client.sh del_task_notif {jobId} {taskId} {notificationId}
```

Corresponds to the "unregisterTaskNotification" method.

```
[piontek@druid-bis bin]$ ./ws_client.sh del_task_notif 1107243045980_ps_ext_3879 taskid_1 \  
1107243045980_ps_ext_3879:taskid_1:1107246248349  
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek  
- Service URL: http://druid-bis.man.poznan.pl:8443/axis/services/grms  
- Notification listener was unregistered successfully.
```

## add\_output

```
./ws_client.sh add_output {jobId} {taskId} {name} {path} {PHYSICAL|LOGICAL}  
{FILE|DIRECTORY}
```

Corresponds to the "addTaskOutputFileDirs" method.

```
[piontek@druid bin]$ ./ws_client.sh add_output 1088499660886_appid_1622 taskid_1 param_file \  
gsiftp://ragel.man.poznan.pl/~examples/param_file physical file  
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek  
- Service URL: http://druid-bis.man.poznan.pl:8443/axis/services/grms  
- The operation finished with result:  
- errorCode= 0  
- errorMessage= OK
```

If the file or directory to be registered already exists the proper error message is displayed.

```
[piontek@druid bin]$ ./ws_client.sh add_output 1088499660886_appid_1622 taskid_1 param_file \  
gsiftp://ragel.man.poznan.pl/~examples/param_file physical file  
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek  
- Service URL: http://druid-bis.man.poznan.pl:8443/axis/services/grms  
- The operation finished with result:  
- errorCode= 420  
- errorMessage= File/Dir already exists
```



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

```
[piontek@druid bin]$ ./ws_client.sh add_output 1088499660886_appid_1622 taskid_1 "" \
    gsiftp://ragel.man.poznan.pl/~examples/param_file physical file
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: http://druid-bis.man.poznan.pl:8443/axis/services/grms
- The operation finished with result:
- errorCode= 422
- errorMessage= Incorrect definition
```

## get\_output

```
./ws_client.sh get_output {jobId} {taskId}
```

Corresponds to the "getTaskOutputFileDirs" method.

```
[piontek@druid bin]$ ./ws_client.sh get_output 1088499660886_appid_1622 taskid_1
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: http://druid-bis.man.poznan.pl:8443/axis/services/grms
- Files/Directories were got successfully.
- Number of File/Dirs: 1
- FileDir[0]:
- Name = param_file
- Path = gsiftp://ragel.man.poznan.pl/~examples/param_file
- UrlType = PHYSICAL
- PathType = FILE
```

## del\_output

```
./ws_client.sh del_output {jobId} {taskId} {name} {path} {PHYSICAL|LOGICAL}
{FILE|DIRECTORY}
```

Corresponds to the "deleteTaskOutputFileDirs" method.

```
[piontek@druid bin]$ ./ws_client.sh del_output 1088499660886_appid_1622 taskId\
    param_file gsiftp://ragel.man.poznan.pl/~examples/param_file \
    physical file
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: http://ragel.man.poznan.pl:8443/axis/services/grms
- The operation finished with result:
- errorCode= 0
- errorMessage= OK
```

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

```
[piontek@druid bin]$ ./ws_client.sh del_output 1088499660886_appid_1622 taskid_1 param_file \
    gsiftp://ragel.man.poznan.pl/~examples/param_file physical file
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: http://ragel.man.poznan.pl:8443/axis/services/grms
- The operation finished with result:
- errorCode= 421
- errorMessage= No such file or directory
```

## add\_checkpoint

```
./ws_client.sh add_checkpoint {jobId} {taskId} {name} {path} {PHYSICAL|LOGICAL}
{FILE|DIRECTORY}
```

Corresponds to the "addTaskCheckpointFileDirs" method. For details please see: the section called "add\_output"

## get\_checkpoint

```
./ws_client.sh get_checkpoint {jobId} {taskId}
```

Corresponds to "getTaskCheckpointFileDirs" method. For details please see: the section called "get\_output"

## del\_checkpoint

```
./ws_client.sh del_checkpoint {jobId} {taskId} {name} {path} {PHYSICAL|LOGICAL}
{FILE|DIRECTORY}
```



Corresponds to the "deleteTaskCheckpointFileDirs" method. For details please see: the section called "del\_output"

### extend\_execution

```
./ws_client.sh extend_execution {jobId} {taskId} {duration}
```

Corresponds to the "extendTaskExecutionTime" method.

```
[piontek@druid bin]$ ./ws_client.sh extend_execution 1088499660886_appid_1622 taskid_1 P0Y1347M0D
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: http://ragel.man.poznan.pl:8443/axis/services/grms
- ExecutionTime extended successfully
```

### description

```
./ws_client.sh description {SHORT|FULL}
```

Corresponds to the "getServiceDescription" method.

```
[piontek@druid bin]$ ./ws_client.sh description short
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: http://druid-bis.man.poznan.pl:8443/axis/services/grms
- GRMS Service version 2.0
```

```
[piontek@druid bin]$ ./ws_client.sh description full
- Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
- Service URL: http://druid-bis.man.poznan.pl:8443/axis/services/grms
- GRMS Service version 2.0
```

```
Service host name: druid-bis.man.poznan.pl
Client host name: druid.man.poznan.pl
Your DN: /C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
Your Proxy: WAS DELEGATED SUCCESSFULLY
```

```
GrmsJobIdentifier submitJob( GrmsJobDescription jobDescription) throws GrmsSubmitJobException
.
.
void extendTaskExecutionTime( GrmsJobIdentifier jobId, GrmsTaskIdentifier taskId, String duration) \
    throws GrmsExtendTaskExecutionTimeException

GrmsJobInformation {
    GrmsProjectIdentifier project;
    String userDn;
    GrmsJobStatusType status;
    Calendar submissionTime;
    Calendar finishTime;
    String errorDescription;
    GrmsTaskIdentifier[] tasksIdentifiers;
    int tasksCount;
    GrmsJobDescription jobDescription;
}

GrmsTaskInformation {
    GrmsTaskStatusType status;
    Calendar submissionTime;
    Calendar finishTime;
    String requestStatus;
    int reqNumStatus;
    String errorDescription;
    GrmsTaskHistory lastHistory;
    int historyLength;
}

.
.
.
```