

# Middleware Prototype – Working Document

## \*\*\*DRAFT\*\*\*

---

Document identifier: XXX-XXX-XX-XXXX

Date:

Contributors: **Predrag Buncic (CERN), Steve Fisher (RAL), David Groep (NIKHEF), Frederic Hemmer (CERN), Peter Kunszt (CERN), Erwin Laure (CERN), Miron Livny (VDT), Francesco Prelz (INFN)**

Editors: **Erwin Laure**

Document status: **DRAFT v0.17**

---

**Abstract:** This working document is used to break down the high level services defined by ARDA to actual components and tries to define the initial set of services provided by the middleware prototype, their interfaces, as well as the technology/systems exploited. A separate document maps these components to existing implementations coming from Alien, EDG, and VDT.

The structure and initial AliEn input is taken from Chapter 5 of (the unpublished) Draft v0.2 of the ARDA document.

Issue	Date	Comment
0.1	08-Dec-2003	First version (E.L.)
0.2	16-Dec-2003	Added more recent AliEn description and general description of services from ARDA document (P.B.)
0.3	17-Dec-2003	Added replica management description from Peter K. (E.L.)
0.4	18-Dec-2003	Added R-GMA description from Steve F. (E.L.)
0.5	19-Dec-2003	Re-ordered sections (E.L.)
0.6	06-Jan-2004	Updated information section (SMF)
0.7	12-Jan-2004	Added description of AliEn I/O (P.B.)
0.8	12-Jan-2004	Added contributions from Francesco Prelz on the “WP1” Network Server interface and the CE functionality that is obtained via CondorG/GRAM (E.L.)
0.9	16-Feb-2004	Added initial data mgmt API from Peter K. (E.L.)
0.10	23-Feb-2004	Added more input on data mgmt from Peter K. (E.L.)
0.11	23-Feb-2004	Document re-structuring: previous work from Alien/EDG/VDT moved to appendix (E.L.)
0.12	01-Mar-2004	More additions to Storage Elements and Data Mgmt (P.K.)
0.13	03-Mar-2004	APIs and data types for CE and TaskQueue from Francesco including some background information for the annex (E.L.)
0.14	04-Mar-2004	Information system and Monitoring added (S.F.) Appendix moved to sub-document (E.L.) Disclaimer added at the end of section 1.
0.15	04-Mar-2004	Added note on security before Section 3.1 Added note on access service in Section 3.1.2 Added disclaimer to Section 3.4. Added short description of CE in Section 3.10 (E.L.)
0.16	07-Mar-2004	Implemented comments from David Groep, Francesco, and Predrag (E.L.) Removed subdocument (doesn't work with office XP) Document restructuring: removal of images; ACLs, APIs moved to appendix Update of information & monitoring section (UK)
0.17	14-Apr-2004	Update of authentication & authorization sections (David Groep) Some comments from Cal Loomis and David Adams applied (comments not yet applied added as comment to the document).

# CONTENT

<b>1</b>	<b>INTRODUCTION .....</b>	<b>2</b>
<b>2</b>	<b>TERMINOLOGY .....</b>	<b>2</b>
2.1	API DEFINITION .....	2
2.2	CONVENTIONS FOR DESCRIBING METHODS .....	2
2.3	ERROR HANDLING.....	2
<b>3</b>	<b>GENERAL DECOMPOSITION .....</b>	<b>2</b>
3.1	API AND GRID ACCESS SERVICE .....	2
3.2	.....	2
3.2.1	<i>Data Services API.....</i>	2
3.2.2	<i>Initial Components used for Prototype.....</i>	2
3.3	AUTHENTICATION.....	2
3.3.1	<i>Privacy preservation .....</i>	2
3.3.2	<i>Initial Components used for Prototype.....</i>	2
3.4	AUTHORISATION SERVICE .....	2
3.4.1	<i>An authentication and authorisation use model.....</i>	2
3.4.2	<i>Enforcement.....</i>	2
3.4.3	<i>Initial Components used for Prototype.....</i>	2
3.5	INFORMATION AND MONITORING SERVICES .....	2
3.5.1	<i>Producer services.....</i>	2
3.5.2	<i>Consumer Service.....</i>	2
3.5.3	<i>Resources.....</i>	2
3.5.4	<i>Security .....</i>	2
3.5.5	<i>Example code.....</i>	2
3.5.6	<i>Initial Components used for Prototype.....</i>	2
3.6	JOB MONITORING .....	2
3.7	JOB PROVENANCE.....	2
3.8	AUDITING SERVICE .....	2
3.9	ACCOUNTING SERVICE .....	2
3.10	SITE PROXY (WAS SITE GATEKEEPER) .....	2
3.11	COMPUTING ELEMENTS.....	2
3.11.1	<i>Initial Components used for Prototype.....</i>	2
3.12	WORKLOAD MANAGEMENT .....	2
3.12.1	<i>Initial Components used for Prototype.....</i>	2
3.13	DATA SERVICES.....	2
3.14	STORAGE ELEMENTS .....	2
3.14.1	<i>Concepts.....</i>	2
3.14.2	<i>Interface Overview.....</i>	2
3.14.3	<i>Storage Resource Management API.....</i>	2
3.14.4	<i>Posix-like File I/O.....</i>	2
3.14.5	<i>Initial Components used for Prototype.....</i>	2
3.15	DATA SCHEDULING .....	2
3.15.1	<i>Transfer Wire Protocol.....</i>	2
3.15.2	<i>File Transfer Service.....</i>	2
3.15.3	<i>File Placement Service .....</i>	2
3.16	FILE AND REPLICA CATALOG .....	2
3.16.1	<i>Concepts.....</i>	2
3.16.2	<i>Functional concepts.....</i>	2
3.16.3	<i>Issues, Discussion .....</i>	2
3.16.4	<i>Initial Components used for Prototype.....</i>	2
3.17	METADATA CATALOG .....	2
3.17.1	<i>File-based Metadata API.....</i>	2
3.18	PACKAGE MANAGER .....	2

<b>4 REFERENCES.....</b>	<b>2</b>
<b>API DEFINITIONS .....</b>	<b>2</b>
API AND GRID ACCESS SERVICE .....	2
INFORMATION AND MONITORING SERVICES .....	2
COMPUTING ELEMENT .....	2
WORKLOAD MANAGEMENT.....	2
STORAGE ELEMENT.....	2
FILE TRANSFER SERVICE .....	2
FILE PLACEMENT SERVICE.....	2
FILE AND REPLICA CATALOG .....	2
<b>POSIX ACLS.....</b>	<b>2</b>
ACL TYPES .....	2
ACL ENTRIES .....	2
VALID ACLS .....	2
CORRESPONDENCE BETWEEN ACL ENTRIES AND FILE PERMISSIONS .....	2
OBJECT CREATION AND DEFAULT ACLS .....	2
ACCESS CHECK ALGORITHM .....	2
ACL TEXT FORMS .....	2
RATIONALE .....	2
CHANGES TO THE FILE UTILITIES .....	2
STANDARDS.....	2

# 1 INTRODUCTION

During the ARDA workshop held at CERN on Dec. 3<sup>rd</sup>/4<sup>th</sup> 2003 it was decided to use the component breakdown and its mapping to AliEn contained in Draft v0.2 of the ARDA document as the working basis for developing a concrete component description and implementation recommendation for a middleware prototype that can be used by the ARDA project to implement end-to-end analysis systems.

The attendees of the ARDA workshop were:

- Predrag Buncic (AliEn)
- Miron Livny (Wisconsin/VDT)
- Francesco Prelz (INFN)
- Torre Wenaus (LCG AA)
- Peter Kunszt (CERN)
- Frederic Hemmer (CERN)
- Erwin Laure (CERN)
- Steve Fisher (CLRC)

Figure 1 shows the components described in the ARDA document:

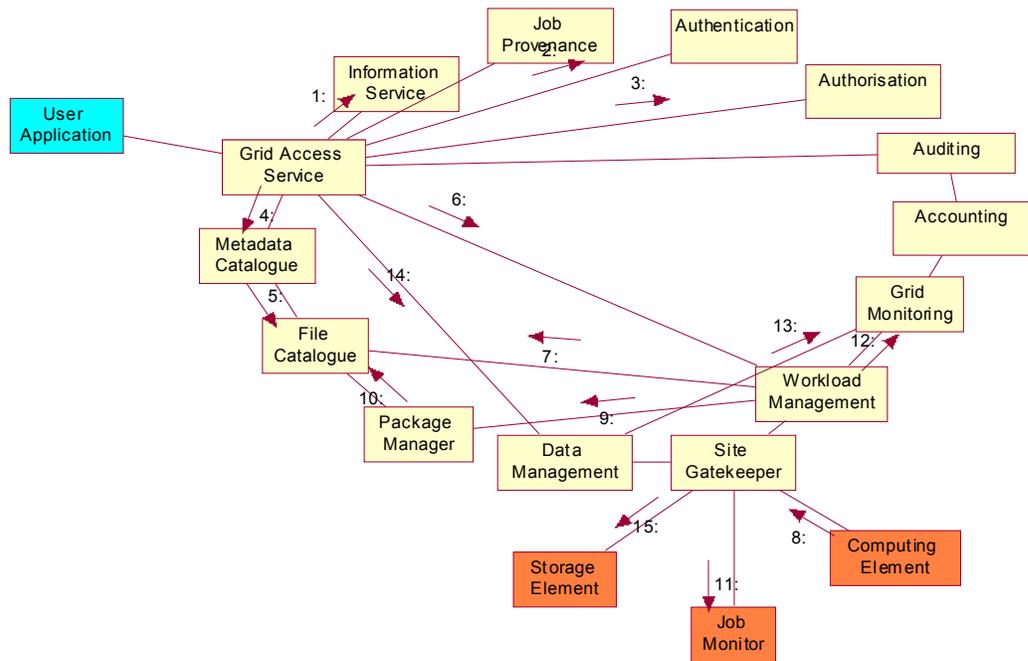


Figure 1: The interaction diagram of key Grid components for typical analysis use case

In the remainder of this document we detail the semantics of these services, derive an initial architecture and design of the services and their interplay, and present an initial API definition.

**Disclaimer:**

**This is a draft working document – many details reported in here are still open for discussion. The initial discussions centred around the CE, SE, and file and replica catalogs.**

## 2 TERMINOLOGY

The terminology listed here is a suggestion to be adapted throughout the ARDA prototype. The acronyms and names suggested below have been suggested through HEPICAL and HEPICAL-II or emerged as the established names in the Grid community in the last couple of years.

Logical File Name	LFN	Unique human-readable identifier of a Grid file.
Global Unique ID	GUID	Unique identifier by construction for a file. Think of it as an inode.
Storage Resource Manager	SRM	A service providing a management interface to mass storage.
Storage URL	SURL	URI of a file on an SRM or at generic storage. The URI has the scheme 'srm' for files managed by an SRM and 'sfn' for other files.
Site		A Grid Site is an administrative domain providing computing and storage resources.
Virtual Organization	VO	A set of Grid users characterized by common usage and access capabilities. Every Grid user belongs to at least one VO.
Storage Element	SE	An SE provides storage to the Grid users. It usually has an SRM interface, but it can also be a simple FTP server or a SAN.
Catalog		A catalog is a collection of data that is updateable and transactional.
Dataset	DS	A dataset is a read-only collection of data.

At this point the suggestion is to introduce an explicit distinction between APIs intended for the human Grid user and for the Grid middleware 'user'. Humans have very different semantic expectations of APIs than automated services, so the exposed APIs are different and have different reasons to exist.

User-Domain API	The API intended for the human user. This includes the API provided to the application programmer, i.e. all high-level application specific tools and interactive processes should be implemented using this API.
Middleware-Domain API	The middleware at this point is being defined as grid services in the same grid layer, providing services to the end user by coordinating low-level grid services and basic computing resources. The Middleware domain API of a low-level service is being used by other middleware to achieve its higher-level task.
Admin API	This is also intended for the human user, but restricted to administrators. It is often useful to explicitly specify that certain APIs are only for administrative usage.

## 2.1 API DEFINITION

In the API definitions below a set of needed functionalities and the way they are used in the scope of a user and administrator of a Grid Service are defined. Since this specification pertains to the usable prototype to be built, we decided to divide interface into three dimensions:

- Structures (objects) to be exposed by the API. Some of the structures must be well defined to preserve semantics throughout the proposal, some may have placeholders to indicate dependence on other services APIs.
- External API exposed to user,
- External Commands - exposed as a command shell, usable by user already in the first iteration of prototyping. Usually wrappers of the equivalent External User API calls or aggregation of these.

Each of these dimensions subsequently covers following areas:

- Regular user operations
- Administrative operations
- Middleware API used usually by other services to be discussed in other documents

## 2.2 CONVENTIONS FOR DESCRIBING METHODS

**Note: the following is not yet consistent in the document; some APIs are still described in java and not using the common prefix; this will be changed in future versions of the document; a WSDL description of the service interfaces will also be provided in a later stage.**

We chose C as the language to describe the API because this is what we expect most of our users to be familiar with. It is straightforward to imagine the analogous API in java and C++ or other languages but this is the subject of a more detailed document once the API is more stable.

Each method is prefixed with `grid_` in order to distinguish them from system calls. This is a C specific notation, for java and C++ name-spacing is more appropriate.

The security model is still not fixed, so some details might change that respect. For the basic methods described here we assume that the user has already authenticated to the service and that the service can enforce the user's access rights based on either the authorization tokens passed, or the identity token used by an external authorization service, or by a context identifier that references an established service security context.

## 2.3 ERROR HANDLING

The error handling is uniform throughout all methods. Each method returns an error number. The error message can be retrieved through the `grid_strerror` method.

- The error handling as described here is specific to C. It is possible to map error numbers and error messages directly into exceptions which is a more common error reporting model for java and C++.
- The error mechanism is that of the unix system calls. Neither AliEn nor EDG have had this concept in their user API, however there have been many requests by the users of EDG to provide these semantics. The advantage is that it is well-known and is a de-facto standard.

The API tables in the rest of the document describe each API, its input value and possible errors. The notes contain arguments why the call was chosen to be included in the specification and how it relates to AliEn and EDG as well as known issues. The table below describes the method to retrieve the error message based on an error number.

<b>Name</b>	<b>grid_strerror</b>
<b>Synopsis</b>	Retrieve string error message for a given error number.
<b>Fields</b>	<code>int errno</code> Error returned by a call <code>char *buf</code> Buffer to place error message in.
<b>Errors</b>	<code>GRID_EINVAL</code> Not a valid error number <code>GRID_ERANGE</code> Buffer not sufficient to store error message
<b>Notes</b>	The behavior is identical to the unix system call <code>strerror</code> . The error numbers and names are prefixed as well to distinguish them from the system errors.

### 3 GENERAL DECOMPOSITION

From the analysis of the AliEn architecture and the services described in the ARDA document, we derive a decomposition of the following key services (as depicted in Figure 1):

- API and corresponding Grid Access Service Component
- Authentication, Authorisation, Accounting and Auditing services
- Workload and Data Management Systems
- File and Metadata Catalogues
- Information service
- Grid and Job Monitoring services
- Storage and Computing elements
- Package Manager and Job provenance service.

In the following we give descriptions of the identified services, pointing out the interfaces they provide as well as potential technologies/systems to be (re)used for the prototype implementation.

In the first phase of the prototype the focus will be on the following services:

- API and Access Service (Section 3.1)
- Authentication and Authorisation (Section 3.3 and 3.4)
- Information Service (Section 3.5)
- Site Gatekeeper (Section 3.10)
- Computing Element (Section 3.11)
- Storage Element (Section 3.14)
- Workload Management (Section 3.12)
- Data Scheduling (Section 3.15)
- File Catalogue (Section 3.16)
- Metadata Catalogue (Section 3.17)

More Services will follow in the future phases of the project.

#### **A note on security:**

It is generally understood that Grid computing requires a security model that allows individual users to access resources based on privileges granted directly to her, granted via membership in one or more Virtual Organizations (VO), or granted via membership in groups within those VOs. For VO based access rights the VO is responsible for assigning group membership to users, but in all cases resource providers must have the ability to trace, and ban, individual users, groups or VOs. Moreover, these authorization decisions must be based on a trustworthy authentication mechanism that allows end-user traceability. It is preferably that the authentication mechanism allows for single sign-on for the users across the widest range of applications. [R17][R18]

Issues regarding authentication and authorization emerge throughout this document in different levels of detail. For example in the use of POSIX-style access control lists in Section 0. Given that security components have not been considered explicitly, interactions may change due to security requirements.

The first version of the prototype will not be able to achieve most of the basic authentication and authorization requirements. Thus it should be realized that adding authentication and authorization might modify components of the API, especially in the initial connect stages.

The initial prototype will thus start from an "opaque VO" model, where users all are VO members, and VO members are all identified by a single common identifier (the AliEn security model). In as far as possible the prototype will allow for the basic requirements to be added later. It is recognized that a model that allows fine-grained authorization and implementation of the authentication and authorization requirements must replace this model.

### 3.1 API AND GRID ACCESS SERVICE

An ARDA API, shown in Figure 2, would be a library of functions used for building client applications like graphical Grid analysis environments, e.g. GANGA or Grid Web portals. The same library can be used by Grid enabled application frameworks to access the functionality of the Grid services discussed in this document. The API is used also to access files available on the Grid as well as to put user's files onto the Grid. By files available on the Grid we understand those stored on one or more Storage Elements and registered in the File Catalogue or replica location service.

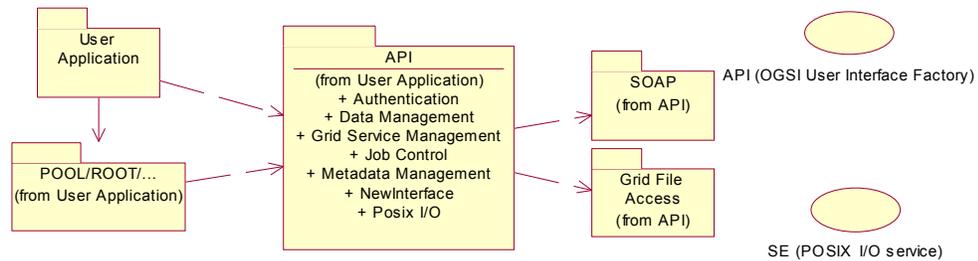


Figure 2: Grid API for user and Grid interactions

The Grid Access Service (GAS) is an example Service Component, and represents the user entry point to a set of core services. When a user starts a Grid session, he establishes a connection with an instance of the GAS created by the GAS Factory for the purpose of this session. The sequence of interactions is illustrated in Figure 3. During the lifetime of the GAS, the user is authenticated and his rights for various Grid operations are checked against the Authorization Service. Thus the GAS is a stateful service that keeps the user credentials and authorization information. Many of the User Interface API functions are simply delegated to the methods of the GAS. In turn many of the GAS functions are delegated to the appropriate service.

### 3.2

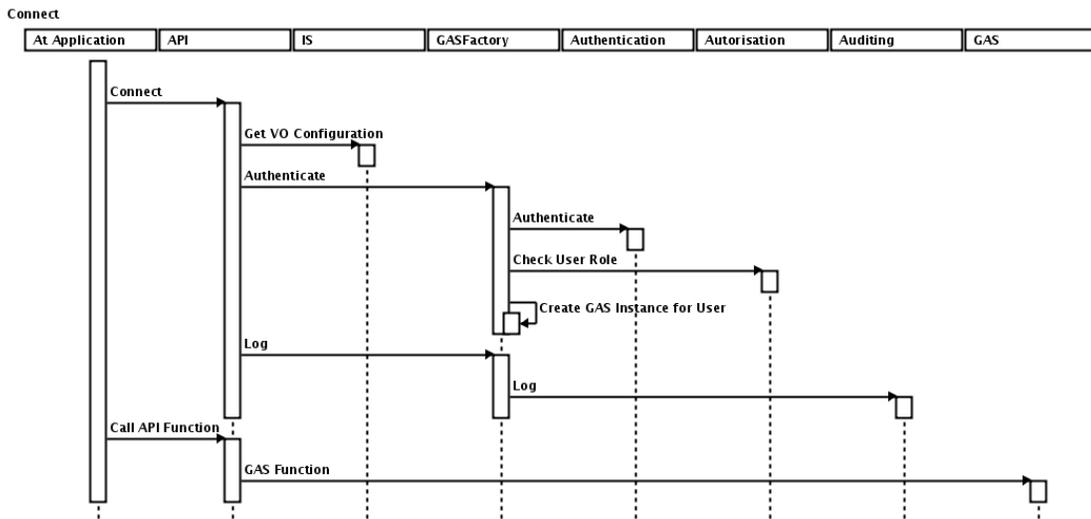


Figure 3: The sequence of interactions between ARDA services while application initiates connection to the Grid

#### 3.2.1 Data Services API

As a part of the ARDA API, the Metadata Catalog, SE, Data Transfer and File Catalogue AP library would expose functions used for building client applications including graphical or interactive Grid analysis environments.

Via the Grid Access Service (GAS) the API is exposed directly or indirectly.

Direct exposure is through client API bindings through java, C, C++ and scripting languages. Indirect exposure is through a unix shell-like interface or eventually through graphical user interfaces, making use of the underlying API.

In the table below there is a list of proposed commands to be implemented in the grid shell [need to specify detailed semantics and error codes].

Command	Description
ls	List the contents of a directory, including metadata of files (through extra options). There should be additional arguments to give the offset and the number of files to return (for scrolling purposes). Should be usable for all kinds of directories (virtual, logical) as well as replicas.
mkdir	Create a new directory
rmdir	Remove an (empty) directory
getfacl	List the ACLs applying to a file or directory
setfacl	Set the ACLs of a file or directory
whereis	Find the location of a file
tree	Print directory tree starting from a given directory. Again give offset and number of entries to return.
cp	Copy files
mv	Rename files

rm	Delete files and replicas
cd	Change directory
Pwd	Print current working directory
Touch	Create an empty LFN
complete	complete the given path (useful for shell completion)
Locate	Find a file in the catalog

### 3.2.2 Initial Components used for Prototype

The access services needs further discussion and will therefore in the first incarnation of the prototype be based on the AliEn shell and a set of APIs.

### 3.3 AUTHENTICATION

Authentication is concerned with identifying entities (users and services) and tying identifiers for these entities to electronic credentials that can be used in establishing secure connections between actors. In the first order approach, an identifier may be the users real name, of the host name of the machine running a service, or an e-mail address. This identifier is bound securely to a piece of data that makes the claim to the identifier non-reputable: the public part of an asymmetric key pair together with the identifier digitally signed by a trusted third party (a “certificate” signed by a “CA”), or a simple password to be memorized by the user, or even a hash of the identifier encrypted with a piece of biometric data.

For simplicity, we will assume in the description of the first prototype authentication based on certificates, but we keep in mind that alternative authentication methods must be supportable in the same scheme. Also, we will assume that the identifier in the credential represents the user identity (foregoing at this time privacy-preserving infrastructures).

The usage pattern for credentials by users and services is slightly different. Services will keep credentials in an un-encrypted form that can be stored locally. Those services that are created on-demand (or on behalf of a user of process on a remote system) should use delegated credentials from either the originating system or responsible user.

On the other hand, users are mobile and their credentials are not only more exposed but also much more “powerful” and attractive to exploitation. Thus, their credentials must be stored securely and be protected by a passphrase.

In either case, the credentials must be issued by a trusted party close to the user (a certification authority, or the user home organisation) and valuable credentials held in a secure storage under the user’s control.

The user will likely have different ways of obtaining credentials, and may have more than one credential at the same time. Thus, a “credential wallet” function should be provided – the infrastructure for this could leverage the MyProxy work [<http://grid.ncsa.uiuc.edu/myproxy/>] to hold these. *In the future, this feature will enable single sign-on for many different applications, including roaming access to wireless networks, etc.*

The MyProxy service provides an API to manage credentials in the credential store. For details of this API the reader is referred to:

<http://www-unix.globus.org/cog/distribution/1.1/api/org/globus/myproxy/MyProxy.html>

### 3.3.1 Privacy preservation

The scenario above will in all cases expose the user's true identity (as testified to by the CA) to all actors. This model is inappropriate for many applications where for example medical or financial data are involved. But end-user tracability must be preserved to satisfy site auditability and incident handling requirements. A straightforward extension of the trusted-third-party model would be to introduce trusted providers of temporary one-time but traceable identity certificates in a format similar to the identifiable credentials. Such "nym" providers should be trusted parties in themselves – operating similarly to the AmEx "one-time credit card number" scheme. They are a special instance of the Identity Providers in the WS-Federation model drafts.

### 3.3.2 Initial Components used for Prototype

- myproxy

## 3.4 AUTHORISATION SERVICE

Authorisation is concerned with allowing or denying access to services to entities based on policies. There are three basic authorisation models [RFC2904], classified as "agent", "push" and "pull". In the Agent model, an authorisation service issues tokens that can be user-held. The user collects the tokens and presents these later to the resource where access is requested. This model is used, e.g., in the VOMS service.

In the Push model the user only interacts with the AuthZ server, and the authZ server forwards service-specific parts of the request to the underlying application service modules (ASMs). Network bandwidth or connectivity provisioning is best done in this mode, since access to the network in the end must be transparent and authZ tokens are hard to negotiate in-band. It is less suited for access to computing elements or storage, since its interface to legacy systems requires pre-configuration of all possible grid users (e.g. in a grid-mapfile).

Lastly, the Pull model expects the resources to be sufficiently clever to contact the relevant AuthZ server and make the necessary decisions. Cellular phone roaming, and various RADIUS-based network access services use this model.

Authorisation is not linked directly to authentication, although identity can of course be used as a basis for authorisation decisions. But in many cases, such decisions can be made without revealing identity (for example when cash payment is involved). And in some cases privacy preservation precludes the use of true identities in authorisation.

The authorisation decisions must be hidden from the user as much as possible (providing single sign-on) and restricted delegation of authorisation should be supported to limit inadvertent damage. The user must be able to hold more than one set of attributes with each identity, and use a combined set of attributes in any authorisation negotiation.

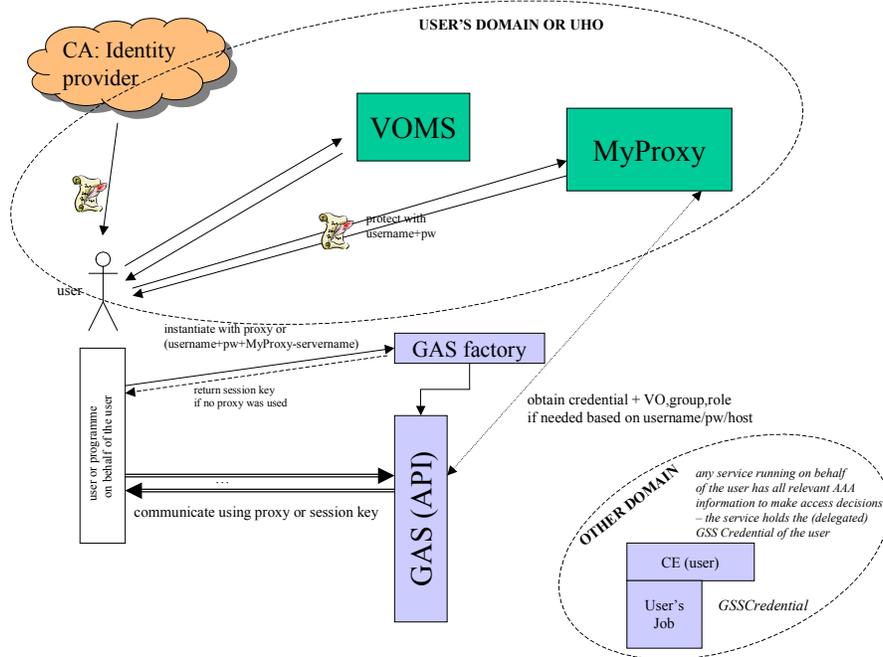
To date, the application of agent-based authZ models seems the most straight-forward way to implement group/role based access to distributed resources. The VO Membership Service (VOMS) [<http://hep-project-grid-scg.web.cern.ch/hep-project-grid-scg/voms.html>] is an attributes issuing service that allows high-level group and capability management and extraction of attributes based on the user's identity. Multiple VOMS attributes can be embedded in the same user proxy credential thus allowing for multiple VO memberships to be asserted in a single call. The VOMS API is documented in <http://edg-wp2.web.cern.ch/edg-wp2/security/voms/apidoc/> VOMS has both an administrative interface for managing VOs as well as a request, extraction and an audit interface.

Once the user has delegated the relevant tokens to the service, it is up to the service to retain and propagate this information along to other services as needed. As long as the credential and authorisation information is retained – for example in a GSS Credential type [RFC2743]. A GSS credential contains all the necessary cryptographic information to enable the creation of a context on behalf of the entity that it represents. It may contain multiple, distinct, mechanism specific credential elements, each containing information for a specific security mechanism, but all referring to the same entity. A credential may be used to perform context initiation, acceptance, or both [JAVA 1.5.0 documentation for GSSCredential, <http://java.sun.com/j2se/1.5.0/docs/api/org/ietf/jgss/GSSCredential.html>].

Although services co-located in the same service container or in the same administrative domain could communicate without using the user’s credential, it is understood that sufficient information must be transferred or delegated along to new services to be able to establish new secure connections. Thus, a GSS Credential representing the user’s (assumed) identity, and authorisation attributes must be passed on service invocations.

### 3.4.1 An authentication and authorisation use model

A high-level view of the control flow in the Access service creation is presented in the figure below. The identity provider (CA) is a trusted third party external to the VO, and also the credential wallet (MyProxy service) should be hosted by a user-trusted entity (like the user’s home organisation or UHO).

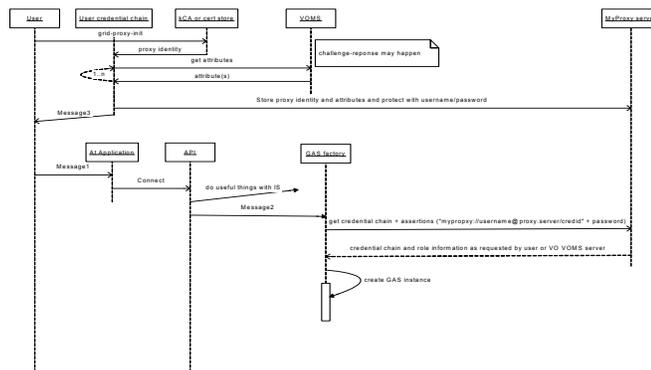


1. The user hold either a long-lived certificate, or is able to obtain a proxy from a Site-Integrated Proxy Service (SIPS). This step concludes the authentication proper.

2. Based on this credential, the user accesses a VO membership service (VOMS) and obtains one or more attribute certificates with embedded group and role information. The VOMS service inspects the user's identity and based on this authentication information signs a membership attribute bound to that identity. These attributes are bound to the user identity. The enhanced user proxy should contain at least one VO attribute.
3. The VO-enhanced proxy is stored in an on-line credential service (MyProxy) and protected with a username/password combination. The username/password may be a one-time combination. The MyProxy service will release the credential to any client that provides the username/password combination (or the original user proxy).

The VO will administer the VOMS database and add the relevant group and role information to this database. Two other points are worth noting:

- the user can add any number of VO attributes (roles, group, even from different VOs) to a single credential chain. This allows the user to make a tradeoff between wide single sign-on and privacy protection (reveal only what's needed)
  - The user can store an infinite amount of different credential chains inside the MyProxy server, protected with different username/password combinations. The user can even use different MyProxy servers for the same VO, by having the hostname of the MyProxy server explicit in the GAS factory API.
  - The password passed to the MyProxy server is (or at least can be) a worthless one-time password protecting only one specific set of credentials and assertions.
4. The end-user (or a programme on his behalf) contacts the access service factory to instantiate a new access service. The user may authenticate using the enhanced proxy, or can now provide the username/password combination and the MyProxy service name. The group and role mapping contained in the VOMS attributes determine the roles and capabilities on the Access Service.



5. In case the username/password combination is used to access the GAS Factory, it could decide on any authentication mechanism to secure the connection between the user (-programme) and the GAS. It can be the symmetric session key used in the TLS communications, or a magic cookie, for instance. In case the user proxy is used no additional token needs to be exchanged for secure communications.

6. To communicate with the GAS API after the initial connection, either the magic cookie or the proxy is used. *In a first prototype using the proxy itself is the most straightforward implementation.*

*The user program is now running and will access grid services continuously. Some of these interactions will be with service 'clusters' that are running as a collection in a single domain. In other cases, the interaction be involve starting remote processes, running jobs, or delegating capabilities to other entities.*

7. The AS will contact other services on behalf of the user program. In such communications, the user credentials could be provided to the service if that service is running outside the domain. If a trusted domain can be defined amongst a set of services, the communications between them could of course use any other mechanism to convey the user's capabilities and roles

### 3.4.2 Enforcement

On the service end the validity of the user credentials and the attributed contained therein must be validated at least once per administrative domain. To this end, both the authentication information must be checked (using the GSS-API, taking into account the newly proposed PKIX extensions for proxy certificates as defined in GSI), as well as the authorisation information.

The basic validation of the authentication data is supported either by the Globus `gss_assist` library or by the GACL slash-grid proxy library at <http://www.gridpp.ac.uk/gridsite/slashgrid/> for C and C++ implementations. For Java, the Java COG [<http://www.globus.org/cog/java/>] or alternatively the EDG Java Security packages are available.

Their API are described in the references quoted above.

The implementation of the policy enforcement points for authorisation is also available for both the native and java-hosted systems, although the usefulness of the policy enforcement in the end highly depends on the use made of the authorisation information inside the service.

For native implementations, VOMS attributed can be parsed and enforced by LCAS [<http://www.dutchgrid.nl/DataGrid/wp4/lcas/edg-lcas-1.1/>] for high-level decisions and access control lists, and by LCMAPS for running executables natively with per-user unix credentials [[http://www.dutchgrid.nl/DataGrid/wp4/lcmaps/edg-lcmaps\\_gcc3\\_2\\_2-0.0.23/](http://www.dutchgrid.nl/DataGrid/wp4/lcmaps/edg-lcmaps_gcc3_2_2-0.0.23/)]. The latter requires the use of a 'sudo' like mechanism that is currently only implemented using the `edg-gatekeeper` (see <http://www.dutchgrid.nl/DataGrid/wp4/lcas/edg-lcas-1.1/node3.html>).

For Java, the EDG Java Authorisation system is available with similar role-mapping functionality for hosted environments. The details and API are given in <http://edg-wp2.web.cern.ch/edg-wp2/security/edg-java-security.html>

### 3.4.3 Initial Components used for Prototype

- VOMS

## 3.5 INFORMATION AND MONITORING SERVICES

The information services are a vital component of any grid; most services will publish or consume information. Some services may add behaviour to the information services but more commonly they will merely use them. These services may choose to hide the underlying information service, but this has the great disadvantage that it is then hard to combine information from the different services. It is best to start with a single well defined interface to the information system and only specialise when it proves necessary. It may be useful to provide simpler APIs for some purposes.

Any information can be monitored provided it carries a timestamp. The mechanisms to move the information around are the same. What makes monitoring systems distinctive is normally the GUIs that are provided to visualise time-sequenced data and to highlight problems. These GUIs are simply clients of the information services.

Some information of interest changes rapidly and some much more slowly. However, even with the slowly changing information, it is often necessary to know quickly if it does change. Publishing information that is only changing infrequently, along with rapidly changing information is inefficient. This requires thought when designing schemas. It is better to treat the information as two or more entities with one to one relationships between them at any one time, rather than trying to bundle together slowly and rapidly changing quantities.

We can consider some of the areas where information services appear:

1. There is the “MDS-like” information service used to publish information about available services.
2. The Job Provenance Service, which keeps track of the execution conditions for all the Grid jobs, could be implemented on top of the information service.
3. An Auditing Service provides the mechanism for all services to report their status and error conditions. This allows Grid manager to monitor all exceptions in the system and to take corrective action. This is simply publishing information using the information service.
4. An Accounting Service could be defined to accumulate information about the use of the Grid resources by the users and groups of users. Again this does not appear to require additional functionality beyond that expected of an information service; however, it requires fine-grained access control and trusted transmission channels.
5. The Package Manager service gives information on the package names, versions and their locations in data repositories, usually Storage Elements. This is once more just publishing information – though perhaps a package manager might also actually manage the packages as the name indicates.
6. Application monitoring and bookkeeping. Applications being executed as part of large production runs can publish their status allowing production coordinators to keep things under control.

R-GMA provides a Producer and Consumer model for information publishing and retrieval, in accordance with the GMA architecture. R-GMA uses a hidden registry to manage subscribed Producers and Consumers. An additional component referred to as the mediator

uses the registry to broker Producer and Consumer lookups using the powerful SQL query language.

The Services currently available within R-GMA are:

- Factory services
  - ProducerFactory service
  - ConsumerFactory service
- Producer services
  - PrimaryProducer service
  - SecondaryProducer service
  - OnDemandProducer service
- Consumer service

### 3.5.1 Producer services

The Producer services are used to publish information into R-GMA. A Producer is created by passing properties to a ProducerFactory service, which creates a Producer and returns a reference so you can interact with it. Currently there are three main types of Producer:

- Primary
- Secondary
- On-demand

These all behave in different ways, as described in the following sections, but have two common features:

#### 3.5.1.1 Declaring tables

A Producer advertises the type of information that it will make available by declaring a table. The user can also specify a predicate (SQL WHERE clause) that defines the precise subset of the table that will be published. If the table has not already been defined, the user must provide a CREATE TABLE statement to describe the columns in the table.

#### 3.5.1.2 ProducerType attribute

The ProducerType attribute specifies the type of data that the Producer will make available.

Producers with the *History* property retain *all* published data (possibly subject to a *minimum retention period*).

Producers with the *Latest* property only hold the most recent tuple for each value of a table's *primary key*. When the user inserts a tuple, one of two things will happen:

1. The tuple will be stored if it is newer than any tuple with the same primary key value in storage or if no such tuple exists.
2. The tuple will be ignored if a tuple with a newer timestamp already exists with the same primary key value.

This enables the Producer to maintain the current status of a monitored component rather than a complete history of events.

#### 3.5.1.3 Primary Producers

Primary Producers are the initial source of the data. They retain data for at least the duration of the *minimum retention period*, which can be set for each table. Tuples older than the

*minimum retention period* are discarded if no Consumers are actively using the Producer. Primary Producers extend the standard Producer in two ways:

#### 3.5.1.3.1 Inserting data

The user introduces new data into R-GMA by inserting it into a Primary Producer.

#### 3.5.1.3.2 Persistency (or storage location)

Non-persistent Producers are created by specifying a *Memory* storage location and are implemented using an in-memory buffer maintained by the Producer service. Tuples are stored within this buffer and dispersed to all listening Consumers. If the system should crash, the data will be lost unless a Consumer has already picked it up. There is a memory-imposed limit on how much data can be stored.

Persistent Producers use an RDBMS to store published data and are created by specifying a *Database* storage location. In this case the Producer can be restarted after crashing and the limit on data storage depends only on how much data the RDBMS can cope with. The user can specify the RDBMS to be used.

#### 3.5.1.4 Secondary Producers

A Secondary Producer is used to aggregate streams of data and/or make them persistent. They do this by setting up a Consumer to retrieve data for each declared table and republishing this through a Primary Producer. Secondary Producers are created by specifying the producer type and persistency attributes of the Primary Producer that will be used to republish the data.

As the user cannot directly publish new data using a Secondary Producer, it has no *insert* operation.

The Secondary Producers has several uses within R-GMA:

1. It collects and maintains information in one place. This avoids the querying Producers individually or accessing remote information sources.
2. It can be set up to record historical data and thereby act as an archiver of information.
3. It offers an alternative to the mediator for joins over tables held in different places.

#### 3.5.1.5 On Demand Producers

An On Demand Producer interacts with a user-supplied plug-in that returns data in response to an SQL query. It is used when the cost of creating messages is high. In this case, efficiency can be greatly improved by retrieving data only when it is requested by a Consumer, rather than retrieving it periodically and publishing it into R-GMA.

### 3.5.2 Consumer Service

The Consumer Service is used to obtain data published by one of the Producer services. A Consumer handles a single query, expressed as an SQL SELECT statement. The Consumer also identifies how the query is executed – referred to as the query type. Depending upon the query type, the Consumer will run its query in one of two ways. The first is called a *Continuous* query and the second is referred to as a *One-Time* query.

*Continuous* queries stream data from Producers. When a new tuple is published, the tuple is copied to all interested Consumers. This approach allows a Consumer to keep up-to-date with all Producer events. Conversely, *one-time* queries involve a single request/response for the Consumer to get information from a Producer. This allows the Consumer to retrieve a historical or latest view of information from a Producer.

An overview of query types is shown in table 1:

Query type	Query mode	Description	Producers that answer the query
CONTINUOUS	Continuous	Consumer initiates continuous request. Tuples are then broadcast to each listening Consumer once published.	All
CONTINUOUS+TIME INTERVAL	Continuous	Consumer initiates continuous request. Producer will stream data that was published within the specified time interval.	All
HISTORY	One Time	Consumer issues a query. Producer responds by executing the query and returning the results.	Producers of ProducerType HISTORY
LATEST	One Time	Consumer issues a query. Producer responds by executing the query and returning the results.	Producers of ProducerType LATEST

**Table 1.** Query Types.

When using either a continuous or one-time query approach, retrieved tuples are stored within a Consumer buffer, managed by the Consumer service. The Consumer can extract buffered data by invoking the various *pop* operations available in the Consumer API.

### 3.5.2.1 Global and local queries

R-GMA makes the distinction between *global* and *local* queries. A global query is a query that is posed against the (virtual) global schema. A local query is a query posed against the (real) local schema of one or more Producers.

The task of a mediator involves translating global queries into a set of local queries, and users normally choose to take advantage of this. However, R-GMA also allows users to send local queries to individual Producers directly. To pose a local query, the user passes a list of Producers to a Consumer. This query is forwarded on to each Producer in the list and the results are merged.

As most Producers are supported by databases, expressive local *one-time* queries can be posed. However, local queries that involve distributed query processing, for example joining a relation of one Producer with a relation of another, are not supported.

### 3.5.3 Resources

What we refer to as a Producer or a Consumer really consists of two parts: an API instance on the user's machine, which allows the user to interact with the Producer/Consumer, and a Resource on the server, which contains the functionality. A Resource is created when the user creates a new API instance using a Factory and can be explicitly destroyed by the user or allowed to "time out" using a system of *soft-state registration*. Each resource has a *termination interval* associated with it, which is the length of time a user can be inactive before the resource is destroyed.

### 3.5.4 Security

The basic authentication mechanism will protect the system from those without an approved certificate. Protection from malicious certificate owners, depends upon internal mechanisms within R-GMA (e.g. mutual authentication between the internal components), this has already

been implemented. The section below describes part of the authorization scheme that will be implemented – though not in the first version of the ARDA prototype.

### 3.5.4.1 Authorization

Authorization rules, which are local to a VO, will define what actions an individual (certificate holder) may carry out. This includes the ability to publish information (via a Producer), to query (via a Consumer) or to discover what Producers exist.

The authorization rules are specified when a table description is first defined in the schema. Once it has been defined, neither the table description nor the authorization rules may be modified, so if either is wrong, the table must be dropped and recreated. The authorization rules are defined in a TableAuthorization object that is passed into the createTable method. This holds a set of rules of the form:

```
View : AllowedCredentials;
```

Each View defines a view on a table in the form of a SELECT statement. If you match the allowed credentials you will have read access to the data defined in that view. It is possible that your credentials match two rules in which case you will be able to see the union of the two views. If you issue a query to see data you are not allowed to see, you will just receive an empty set.

Both the View and the AllowedCredentials are parameterised. The keywords DN, VO, GROUP, ROLE and CAPABILITY may be replaced by their actual values. This is indicated by enclosing them in [].

If the TableAuthorization object contains no rules, everyone has read access.

Consider a table created as:

```
CREATE Table Job (Jobid VARCHAR(200), State VARCHAR(30),  
Owner VARCHAR(200), OwnersGroup VARCHAR(20), Usage  
VARCHAR(50), JobDesc VARCHAR(200))
```

To impose the constraints that a row of the table is available to the owner of the job, i.e. if the DN matches.

This would be achieved with the following rule:

```
SELECT * from Job where Owner=[DN] : DN=[DN];
```

To allow the VO admin role to see all but the JobDesc field:

```
SELECT JobID, State, Owner, OwnersGroup, Usage from Job :  
ROLE='Voadmin';
```

To allow all members of the ABModel group to have access to information, including the JobDesc, but excluding the usage information we get the statement

```
SELECT JobID, State, Owner, JobDesc FROM Job WHERE  
OwnersGroup='ABModel' : GROUP='ABModel'
```

## 3.5.5 Example code

### 3.5.5.1 Basic Primary Producer

The code below shows you how to create a basic Primary Producer and use it to publish data.

```
try {  
  
    // Create an instance of the producer factory  
    ProducerFactory producerFactory =  
        new ProducerWebServiceFactoryImpl();
```

```

// Create a LATEST DATABASE primary producer
// The list of VO names is omitted, so the producer will publish
// to all VO's it is a member of.
PrimaryProducer primaryProducer =
    producerFactory.createPrimaryProducer(
        StorageLocation.DATABASE, ProducerType.LATEST, null);

// Declare table JobDetails with no predicate (the table must
// already have been created)
primaryProducer.declareTable("JobDetails", new Predicate(""));

// Insert data into the producer
primaryProducer.insert(new InsertStatement(
    "INSERT INTO JobDetails VALUES
('jid293','myJob','Running','myDN')"));

// Close the producer
primaryProducer.close();

} catch (RemoteException e) {
    // The PrimaryProducer or ProducerFactory service could not be
    // contacted
} catch (UnknownResourceException e) {
    // The PrimaryProducer resource could not be found
} catch (RGMAException e) {
    // An application error occurred (e.g. incorrect SQL syntax)
}

```

### 3.5.5.2 Advanced Primary Producer

The code below shows you how to create and use a Primary Producer with the following features:

- a predicate
- a CREATE TABLE statement
- authorization
- a *minimum retention period* for the data of 5 minutes
- a *termination interval* for the Producer of 1 hour
- a run-time decision on which R-GMA implementation to use (just set the variable *producerFactoryImplClassName* to the required implementation class).

```

try {
    // Create an instance of the producer factory, using implementation
    // class producerFactoryImplClassName
    ProducerFactory producerFactory =
        (ProducerFactory)
        Class.forName(producerFactoryImplClassName).newInstance();

    StringList voNames = new StringList();
    voNames.addString("alice");
    voNames.addString("atlas");

    // Create a HISTORY MEMORY primary producer, publishing to
    // VO'S alice and atlas.
    PrimaryProducer primaryProducer =
        producerFactory.createPrimaryProducer(
            StorageLocation.MEMORY, ProducerType.HISTORY, voNames);

    // Change the termination interval to 1 hour (from the default 20
    // minutes)
    primaryProducer.setTerminationInterval(
        new TimeInterval(1, Units.HOURS));

    // Create authorization rules for VO's alice and atlas
    TableAuthorization tableAuthz = new TableAuthorization();
    tableAuthz.addRule(
        "SELECT jobStatus, jobName FROM JobDetails WHERE owner=[DN]:" +
        "ROLE='admin'");

    // Create the table
    primaryProducer.createTable(
        new CreateTableStatement("CREATE TABLE JobDetails (" +
            "jobId VARCHAR(20), jobName VARCHAR(40), " +
            "jobStatus VARCHAR(300), owner VARCHAR(200))",
            tableAuthz)
    );
}

```

```

// Declare table JobDetails
primaryProducer.declareTable("JobDetails",
    new Predicate("WHERE site='RAL'"),
    new TimeInterval(5, Units.MINUTES));

// Insert data into the producer
primaryProducer.insert(new InsertStatement(
    "INSERT INTO JobDetails VALUES
('jid293','myJob','Running','myDN')"));

// Destroy the producer: close it immediately, throw away all data
primaryProducer.destroy();

} catch (RemoteException e) {
// The PrimaryProducer or ProducerFactory service could not be
    contacted
} catch (UnknownResourceException e) {
// The PrimaryProducer resource could not be found
} catch (RGMAException e) {
// An application error occurred (e.g. incorrect SQL syntax)
}

```

### 3.5.5.3 Basic continuous consumer

The code below shows you how to create a Consumer and use it to retrieve data from Producers.

```

try {
// Create an instance of the consumer factory
ConsumerFactory consumerFactory =
    new ConsumerWebServiceFactoryImpl();

// Create a Consumer with a LATEST query
// The list of VO names is omitted, so the consumer will consumer
// from all VO's it is a member of.
Consumer consumer = consumerFactory.createConsumer(
    new SelectStatement("SELECT jobName, jobStatus FROM JobDetails"),
    QueryType.LATEST, null);

// Start the query with a timeout of 2 minutes
consumer.start(new TimeInterval(2, Units.MINUTES));

// Wait for the query to finish executing
while (consumer.isExecuting()) {
    Thread.sleep(2000);
}

// Retrieve data from the Consumer
ResultSet result = consumer.pop();

// For each row of results...
while (result.next()) {
// Get the job name (using the column number, 1)
    String jobName = result.getString(1);
// Get the job status (using the column name)
    String jobStatus = result.getString("jobStatus");
    System.out.println("Name: " + jobName + ", status: " +
        jobStatus);
}

// Close the consumer
consumer.close();

} catch (RemoteException e) {
// The Consumer or ConsumerFactory service could not be contacted
} catch (UnknownResourceException e) {
// The Consumer resource could not be found
} catch (RGMAException e) {
// An application error occurred (e.g. incorrect SQL syntax)
}
}

```

### 3.5.5.4 Secondary producer

The code below shows you how to create a Secondary Producer to republish data to persistent (DATABASE) storage.

```

try {
    // Create an instance of the producer factory
    ProducerFactory producerFactory =
        new ProducerWebServiceFactoryImpl();

    // Create a secondary producer
    SecondaryProducer secondaryProducer =
        producerFactory.createSecondaryProducer(
            StorageLocation.DATABASE, ProducerType.LATEST, null);

    // Declare table JobDetails with no predicate.
    secondaryProducer.declareTable("JobDetails", new Predicate(""));

    // Send a "sign of life" to keep the producer alive every 10 minutes,
    // otherwise the SecondaryProducer will close itself after the 20
    // minute default termination interval has expired.
    while (!exit) {
        secondaryProducer.showSignOfLife();
        Thread.sleep(10 * 60 * 1000);
    }

    // Close the producer
    secondaryProducer.close();
} catch (RemoteException e) {
    // The SecondaryProducer or ProducerFactory service could not be
    // contacted
} catch (UnknownResourceException e) {
    // The SecondaryProducer resource could not be found
} catch (RGMAException e) {
    // An application error occurred (e.g. incorrect SQL syntax)
}

```

### 3.5.6 Initial Components used for Prototype

- R-GMA

## 3.6 JOB MONITORING

Job Monitoring is a service that wraps up the running job and provides information about job status and progress. Upon request, it presents this information to other services and provides access to the job specification (JDL, etc) as well as to temporary and final files produced by the job (stdout, stderr, log files, other outputs). The Job Monitoring Service communicates with clients outside the Grid site via a Site Gatekeeper running on the gatekeeper node.. The Gatekeeper Service is either a part of the distributed Workload Management Service or an independent service.

**To be added at a later phase of the project.**

## 3.7 JOB PROVENANCE

The Job Provenance Service is a specialized database to keep track of the execution conditions for all the Grid jobs. This information is used to reproduce the execution environment for the verification and debugging purposes and possibly for rerunning certain jobs. The Job Provenance Service does not contain the information used in the data queries.

**To be added at a later phase of the project.**

## 3.8 AUDITING SERVICE

An Auditing Service provides the mechanism for all services to report their status and error conditions. This allows Grid manager to monitor all exceptions in the system and to take corrective action.

**To be added at a later phase of the project.**

### 3.9 ACCOUNTING SERVICE

The Accounting Service accumulates information about the use of the Grid resources by the users and groups of users. This information serves to prepare Grid usage statistics reports. It is used also in the enhanced workload management with quotas and other policies taken into account.

**To be added at a later phase of the project.**

### 3.10 SITE PROXY (WAS SITE GATEKEEPER)

The site gatekeeper is a proxy service that routes messages to/from the worker nodes of a site. It is essential to allow worker nodes that don't have outbound connectivity to communicate with other Grid services. The site gatekeeper may also optimize the requests by e.g. message grouping or suppression of redundant requests.

**To be added at a later phase of the project (albeit with high priority)**

### 3.11 COMPUTING ELEMENTS

Computing Element (CE) is a service representing a computing resource. Its interface should allow execution of a job on the underlying computing facility, access to the job status information as well as job control (removal). The interface should also provide access to the dynamic status of the computing resource like its available capacity, load and number of waiting and running jobs. The status information should be available on per VO basis or each VO allowed to the site has its own instance of the service.

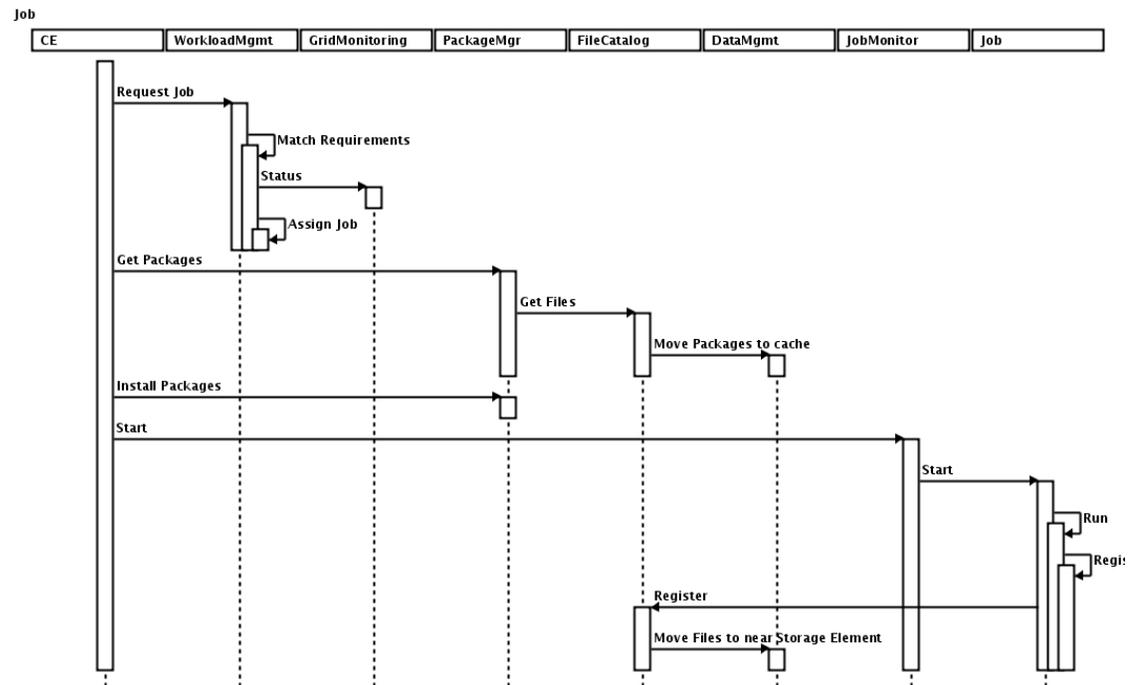


Figure 4: The sequence of interactions between ARDA services illustrating possible job execution model

The computing element is a layered service, which will be able to submit jobs to various batch systems (LSF, PBS, Condor) but also to Grid systems like GT2, GT3, and Unicore. This diversity will be achieved by exploiting CondorG with appropriate backends as queuing system on the CE. This additional queue on the CE offers also the possibility of using the CE in a pull (e.g. the Alien CE pull component may feed this queue) or push (e.g. the EDG resource broker may feed this queue) mode. In order to support security, an appropriate call-out module that changes the job ownership when it is submitted to the local batch system needs to be integrated. An existing component providing this functionality would be for instance the EDG LCAS and LCMAPS modules.

Figure 5 gives a high level overview of the envisaged CE design.

In the first incarnation of the prototype, the pull model via the AlienCE will be pursued. While CondorG will be interfaced to LSF, the direct submission from the AlienCE to LSF will remain enabled.

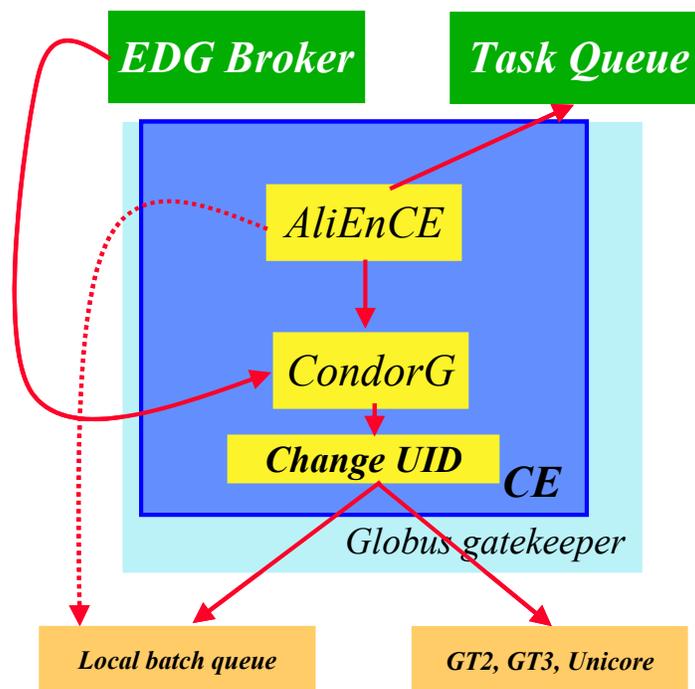


Figure 5: High level schematic overview of CE components and their interaction with workload management modules

### 3.11.1 Initial Components used for Prototype

- Alien CE
- EDG/Globus Gatekeeper
- CondorG
- Change-UID (LCAS, LCMAPS)

## 3.12 WORKLOAD MANAGEMENT

The Workload Management Service (WMS) receives the workload instructions from the users in the form of jobs. It is responsible for assigning jobs to an appropriate Computing Elements (CE) where the job can be run. This can be done via a push model where the job is pushed to an CE or a pull model where the CE is asking for jobs. Initially, the pull model will be provided. If no CE is able to run the job, some preparatory actions can be undertaken, e.g. bringing some of the input data to a near SE. The WMS can modify the job descriptions, e.g. generate subjobs in order to optimise the overall job execution. The WMS assigns an identifier to the accepted jobs that can be used later to interrogate the job status. The WMS provides accounting information upon the job execution to the Accounting Service and can be implemented as a compact or a distributed service, i.e. having internal distributed components

### 3.12.1 Initial Components used for Prototype

- AliEn task queue

## 3.13 DATA SERVICES

This section gives an overview of all data related services. The main services that relate to file access are the File Catalog, Data Scheduling and Storage Element. Metadata Catalogs are mostly also in this area, although this depends on how ‘metadata’ is defined. We will define it through the specification in great detail below. Also of relevance are the Authentication and Authorization as well as the Access Service components. And all services need to provide Auditing and Accounting so they are discussed here as well to a limited extent. The Package Manager might make use of the other service components to achieve its task.

The granularity of the data is on the file level. The idea is to give the user the illusion of a global file system (which is specific to his VO). There may be a client application that can look like a shell (as in AliEn) which can seamlessly navigate in this virtual file system, listing files, changing directories, etc. To read and write files is possible through a posix-like I/O. In terms of access control, the suggestion is to adopt posix ACLs. They have well-defined semantics and are already implemented in several file systems today. The many different flavours of mass storage systems should be hidden behind the very same posix-like file I/O. However, the semantics of reads and write will be affected by having an MSS backend: there may be substantial latencies for reads and many more failure modes for write, so the number of errors and messages is larger than for a conventional file system. The components in the ARDA document covering these issues are the Storage Element (providing the MSS backend abstraction, virtual file system), File Catalog (providing the global logical file system view for the user), and of course the security components (Authentication, Authorization).

In a distributed environment, there will be many replicas (managed copies) of the user’s files stored at different physical locations. The user does not necessarily needs to be aware of this fact, however the capabilities for controlling the replica placement need to be available. This is covered by the Data scheduling component which provides file placement capabilities.

The way the files are found and selected is not necessarily by name but by attributes, and this is where the Metadata Catalog component comes into play. However, this can be very

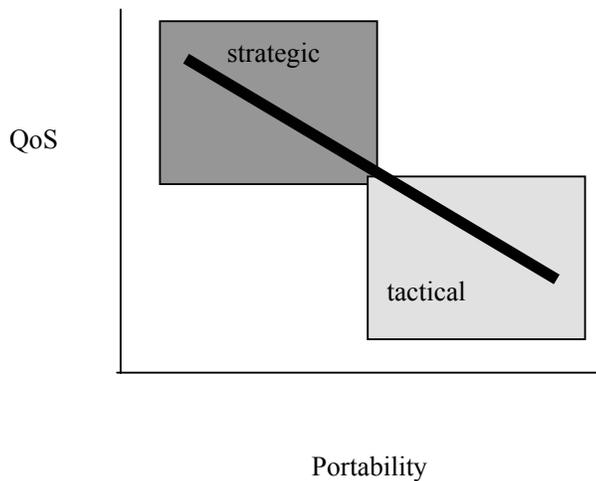
application (and VO) specific, so the interface we specify in this document is a very simple one that can be implemented on top of existing application metadata services such that they can be used from within the grid environment. It is this mechanism that enables the concept of virtual datasets as well

### 3.14 STORAGE ELEMENTS

The Storage Element (SE) is responsible for saving/retrieving files to/from the local storage that can be a disk or a mass storage system. It manages disk space for files and maintains the cache for temporary files. In order to describe the proper service semantics, we need to introduce some concepts that will be applied.

#### 3.14.1 Concepts

Storage Elements represent a resource that may have very different quality of service metrics between different instances (see Figure 6).



**Figure 6 Space occupied by our different storage concepts in the space of QoS and Portability.**

##### 3.14.1.1 Opportunistic, 'tactical' Storage

At one extreme end we have Storage Elements that have a very simple purpose: store grid files close to a computing element as long as it's needed (if possible). Such SEs can be viewed as file caches, temporary or even scratch space. Their value is in their flexibility. Since they do not provide high data safety, such SEs can be easily deployed, switched on or off according to the need of a given site or a given virtual organization.

SEs are controlled by the sites and are subject to local policy. Having the concept of tactical SE allows a site to declare space in a local store in an opportunistic manner. It can provide storage that is currently unused by their local users and revoke it whenever necessary. This will make it attractive to sites to make resources available to grid users, knowing they can reclaim the resources whenever they want. Grid users profit from such storage being able to run jobs requiring local store at more sites. Users are expected to keep only disposable data in such stores – meaning that it should not matter to anyone if the instance of the data is lost – because it can be re-generated or re-copied from a master instance for example. Important data, master copies should not be kept in such storage (only at the user's own risk). If users generate new data at such stores, they should either register a master copy at a more long-term SE or be prepared to re-generate the data if necessary.

The tactical SEs may be hot-deployable and be alive only for a short period of time.

### 3.14.1.2 Fail-safe, 'strategic' Storage

Such storage comes with a high quality of service. Users may expect to be able to reliably retrieve their files from such storage at any later time. Such store usually has a managed MSS behind it and virtual organizations are expected to pay the price of such storage, like the HEP experiments are expected to pay for Castor storage at CERN.

ARDA middleware will provide the tactical SE and will use external tactical or strategic SEs wherever possible and wherever the interface is provided.

### 3.14.1.3 Near storage elements

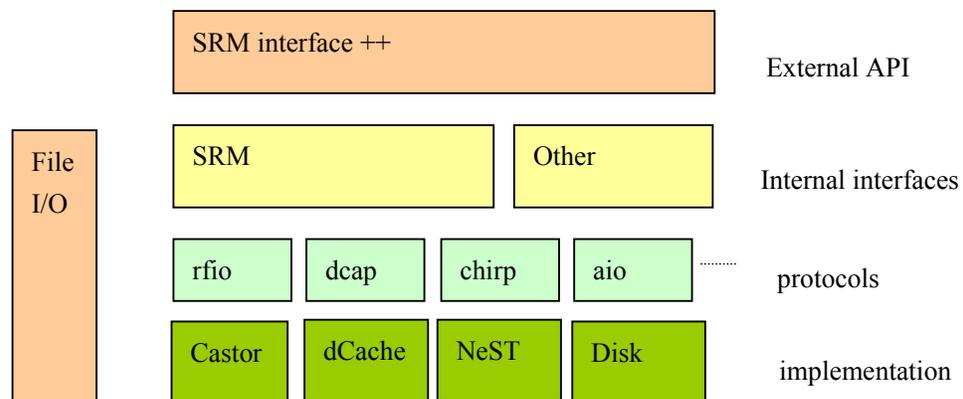
'Near storage elements' are operational storage elements sitting close to the computation. We might use this term for either tactical or strategic SEs. The coupling term 'near' refers to a computing element from which this SE is easily accessible.

### 3.14.1.4 Far storage elements

'Far storage elements' are the opposite of near, but wherever we use this term we think of a strategic, very reliable storage to store master copies or other important data.

## 3.14.2 Interface Overview

There are two interfaces into the SE. The first interface is the SRM interface. It is there to manage the storage space and we foresee to evolve that interface according to the proper SRM specifications, see <http://sdm.lbl.gov/srm-wg/documents.html>. We might extend this interface if we see the need, always keeping in sync with the SRM specification group and trying to work our changes that we feel necessary into the specification.



**Figure 7 The SE modular breakdown. The two interfaces that are exposed are SRM and a posix-like File I/O. These interfaces might make use of many third-party storage and protocol components.**

As displayed in Figure 7 the two interfaces are the SRM management interface and the File I/O interface that exposes a *posix-like* file interface detailed below.

### 3.14.3 Storage Resource Management API

The storage element management interface that we propose to adopt is that of the SRM. It is described in great detail in the documents available at <http://sdm.lbl.gov/srm-wg/documents.html>.

This management API is intended to be used mostly by administrators and as an internal API between Grid Services.

### 3.14.4 Posix-like File I/O

The interaction with the storage element should be transparent to the user through a virtual file system with posix-like semantics. Most probably we have to relax the posix API and implement only a subset, as it is done in I/O libraries today.

It is possible to provide a grid virtual file system such that the system calls can be used for handling files. There are existing approaches to provide such a system, like AlienFS, GFAL, slashgrid, however none of them have been used and deployed in a large scale Grid yet, so an evaluation needs to be done first. There is also doubt about whether it is necessary to provide such a virtual file system at all.

The API specification in the appendix provides the user with a file I/O interface with limited semantics. For the user it should not matter which technology is used by the underlying SE implementation (see Figure 7) as long as the functionality described below is achieved.

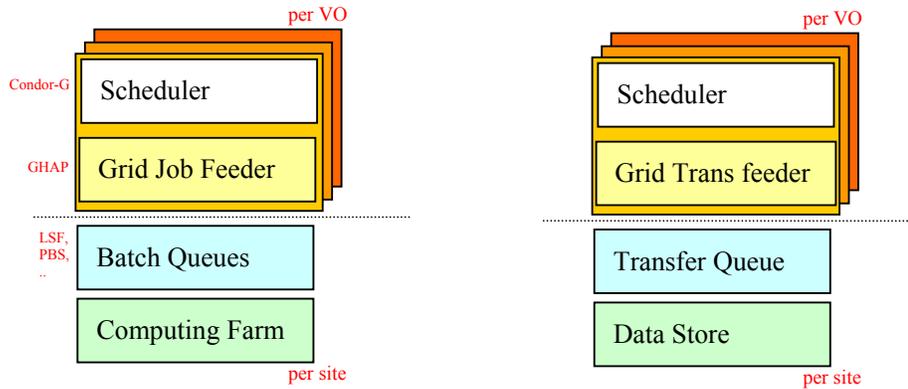
### 3.14.5 Initial Components used for Prototype

- SRM (Castor implementation)
- GridFTP
- GFAL and/or AIOD

## 3.15 DATA SCHEDULING

There is a symmetry between job and data scheduling. The jobs are being run at computing farms that are managed as a local resource by the participating sites. The data are also stored at storage nodes that are managed as a local resource by the very same participating sites. On top of the resource there are the services to distribute jobs and data. These services may be specific to a virtual organization.

In the Grid, replicas are created to make data available to jobs at the many computing sites. These replications may either be scheduled by a user or may be initiated by the job scheduler. Either way, the data transfers need to be controlled through a data scheduling service in order to optimize the network usage and to avoid duplicate transfers. The most obvious scenario that needs to be avoided is the case where hundreds of jobs request the same file to be replicated to the local site, resulting in hundreds of simultaneous file transfers of the same file. In Figure 8 we outline the similarities between the job and data scheduling component stack.



**Figure 8: Possible similarities between job and data scheduling. The left hand side is a simplified Grid Scheduling stack, with the emphasis on the site and VO context. A similar stack can be provided for the data management, where the Data Store is our Storage Element. Today there are existing components that can be evaluated in AliEn (File Transfer Daemon; for the Grid Transfer Feeder) and in Condor (Stork; for the Scheduler).**

The data scheduling stack on the right-hand-side of Figure 8 has been modelled after AliEn (<http://arxiv.org/abs/cs.dc/0306068>). There are possible solutions available for some parts, like Stork from Condor and the AliEn file transfer daemon (FTD).

Also, the underlying resource that is being managed is not only the storage (which actually may have a VO-specific interface to it) but also the network(s) connecting the given site to the WAN, so the right-hand-side of is over-simplified.

### 3.15.1 Transfer Wire Protocol

The actual wire protocol being used to transfer the files is independent of the data scheduling stack. It should be possible to slot in different protocols in a pluggable manner. We will support at least GridFTP as a wire protocol.

### 3.15.2 File Transfer Service

For the initial prototype we plan to provide a File Transfer Service (FTS) and a File Placement Service, explained below. Why do we need the FTS? The current implementation of file transfer in SRM or simply through a GridFTP server (and also RFT) do not provide transfer queues. Also, SRM copy and GridFTP put are not persistent. If the server dies, the copy processes are gone. Globus's Reliable File Transfer service will manage the dying servers but does not deal with 'smart' queuing either, so many processes requesting the same copy are possible.

The Data Transfer service maintains and manages a queue of pending transfers across its allocated bandwidth at a given site. A user can request the replication of a given file from one site to another if he knows the file to exist at a given source and has proper access rights at the given destination. However, if the source is not known, the Data Placement service may find the 'best' source based on network monitoring to be used.

The transfer scheduling and replication methods are described in detail below. The transfer scheduling is based on the AliEn transfer services described below. The replication APIs are based on the EDG replica manager. The File Transfer Service does not include registration in the catalogs. The File Placement Service is responsible for catalog interaction. The File

Placement Service makes use of the File Transfer Service queue internally on behalf of the user.

Each FTS is associated with a site. Just like the SE, the data transfer service should be able to serve more than one virtual organization. It can be argued that the FTS is actually an integral part of a 'Storage Element'.

#### 3.15.2.1 Initial Components used for Prototype

- AliEn FTD

### 3.15.3 File Placement Service

The placement service is a service that will initiate transfer (i.e. make use of the FTS) on the user's behalf. This service API enables the user to create new replica instances. It does also involve registrations in the catalogs. The operations are atomic, however, system failures may leave the catalogs and the actual data content of the SEs involved in an inconsistent state.

#### 3.15.3.1 Initial Components used for Prototype

- EDG replica manager
- AliEn file placement services

## 3.16 FILE AND REPLICA CATALOG

The file catalog is the starting point for file-based data management. In the Grid the user identifies her files by logical file names (LFNs). The LFN is the key by which the Grid services locate the actual replicas of the files. The replicas are identified by SURLs, i.e. each replica has its own SURL, specifying implicitly which SE needs to be contacted to extract the data. Usually, users should not have to deal with SURLs, in all their scope the only names they should need to use are LFNs. The Grid should provide the look and feel of a single file system.

To give this illusion, the Grid data management middleware has to keep track of SURL - LFN mappings in a scalable manner. The Grid File Catalog provides the logical file system view to the user, with all the functionality to group files into directories and to provide access control through posix ACLs. The Grid Replica Catalog provides the mapping of the replicas.

### 3.16.1 Concepts

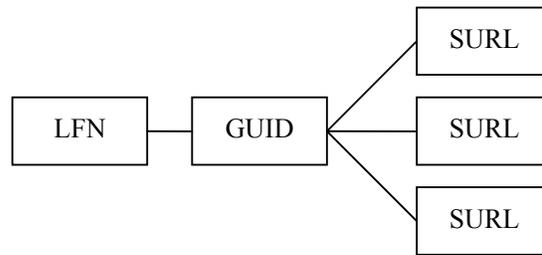
#### 3.16.1.1 Terminology of File Names

We define the terms:

- **SURL Storage URL:** This is a physical instance of a file replica. Also referred to as the Physical File Name (PFN).
- **LFN Logical File Name:** A logical (human readable identifier) for a file. LFNs are mutable, i.e. they can be changed by the user.
- **GUID Globally unique ID:** A logical identifier guaranteed unique by construction, regardless of where it is produced. GUIDs are immutable, i.e. they cannot be changed by the user. Once a file acquires a GUID it must not be changed otherwise consistency cannot be assured. Also, GUIDs are being used by the applications (like POOL) to provide external pointers. If these change, the application will suddenly point to the wrong file.

The mapping we consider is 1 LFN to 1 GUID to many SURL. It is a requirement of the LFN that it be unique. The namespace of the LFN is a directory structure, e.g.

/grid/atlas.lhc.org/production/run/07/123456/calibration/cal/cal-table100



**Figure 9: The mapping model. The introduction of the GUID (which is entirely kept internal) allows to distribute the catalog across the wide area and to catch accidental duplicate LFNs should they occur.**

We define the LFN here to be the primary logical filename for that logical file. This full name must be unique within a VO and should be accessible to the entire VO. Secondary logical references are discussed below. **The internal GUID does not need to be exposed to the users, who will usually only see the human readable LFN.** The purpose of the internal GUID is to allow recovery in the case of clashes where two files are given the same LFN. An example case where this might occur is when a batch farm producing some output is disconnected from the wide area network and registers a new file (and a new LFN) in its Local File Catalog. Upon reconnection, the Local File Catalog tries to resync with the rest of the world, and finds the LFN already registered. The clash can be dealt with by some configurable policy, the easiest of which would be to mail an administrator. The GUID then gives a guaranteed-unique handle that the administrator can use to reference the file while dealing with the clash. The typical resolution would be to assign the file a different LFN. In general, the application should take reasonable steps to ensure that the LFN is unique; the process above is only for recovery purposes.

The SURL is what the storage resource (SE) uses to access the file. In the application discussed here, users should not see the SURL, only the logical filename and its directory structure.

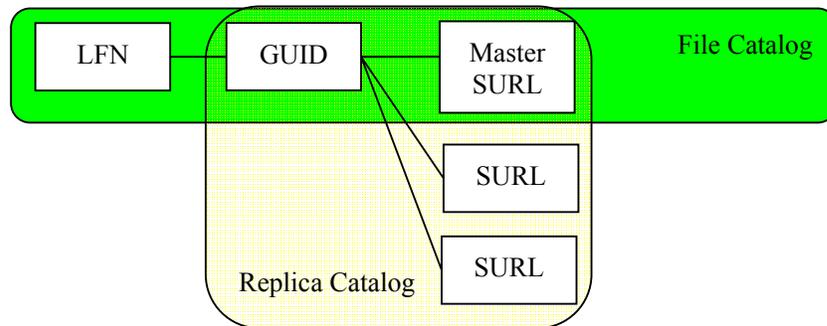
For the logical side, the analogy from the UNIX file system is: The GUID is like the inode. The logical filename is a unique hard link to that inode.

### 3.16.1.2 Master Replica

The master replica is the single one replica where write and update operations are allowed. This is also the master source for replications. If the master replica is lost, it might be recovered from other replicas or not, based on policies. A master replica should always be kept on a ‘strategic’ SE (see 3.14.1.2).

### 3.16.1.3 Catalog Responsibilities

The file and replica catalogs store part of the mappings described in Figure 9. The file and replica catalogs are responsible for part of the mapping space as shown below in Figure 10.



**Figure 10 The responsibilities of the file and replica catalogs.**

The File Catalog may be also a single centrally deployed catalog if the LFN has to be assuredly unique.

### 3.16.2 Functional concepts

We define the concept of a *logical directory*. The directory is just as in a normal file system a list of files and other directories. It can be navigated in the same manner. The full logical filename path contains also implicitly directory structure that can be navigated in such a manner to reference all the logical files for a given VO. The grid middleware dereferences the logical filename to the SURLS, the physical instances of the file. The posix ACL semantics are also enforced through the catalog.

#### 3.16.2.1 Metadata

In the proposed specification the Metadata Catalog module may be directly queried through the File Catalog. The Metadata Catalog has a specific interface which returns a set of LFNs (limited to a maximal size) for a query. The schema and content of the Metadata Catalog needs to be known by the user issuing the query but is not specified by the system. We only specify the generic metadata interface and its error modes. This interface is then used by the File Catalog and is implicitly exposed through the metadata operations, which also involves virtual directories.

#### 3.16.2.2 Logical directory

A new logical directory can be created through a simple API call (mkdir, see below). A user can copy logical files from other logical directories in to their own directory. Symbolic directory links are possible to other directories.

#### 3.16.2.3 Read-only metadata defined virtual directory

This is a logical directory created from a metadata query (see mkdir API section). A user defines a metadata query whose output is a set of logical files that are then made accessible through the virtual directory as symlinks. Only a limited set of operations are available in such directories. We do not foresee the possibility to have subdirectories in virtual directories for example. Also, adding files to a virtual directory is only possible by refreshing the original query.

Virtual directories can be created with the `grid_mvkdir` command by specifying a metadata query. The result set is displayed as an enumerated set of symlinks in the directory. The query can be refreshed explicitly by issuing the `grid_vdir_refresh` command. The contents of the directory do not change before such a refresh operation is issued. Virtual directories have a well-defined maximal size which cannot be exceeded.

### 3.16.2.4 Symlinks

Similarly to symbolic links in Unix, LFN may have a nonzero number of symlinks. Symbolic links however have always be given using absolute paths. It has the same semantics as the unix filesystem symlinks, i.e. they are weak and could point to non-existing LFNs. Any operation on the target LFN does nothing to update the link, symbolic links can create cycles, etc.

### 3.16.2.5 HEPCAL Datasets

The HEPCAL DataSet concept maps to a logical directory which has associated metadata linked to it through the Metadata Catalog. The virtual data concept maps to the virtual directories.

## 3.16.3 Issues, Discussion

### 3.16.3.1 Security

By imposing posix ACLs on the filesystem the security semantics are rather straightforward. This should also help in avoiding concurrency issues when writing into the catalog since each user will have only limited access rights in the LFN namespace and there should be only a finite set of administrators per VO who have full access rights for all of their LFN tree. The probability of two users with the same access rights to write into the catalog in the same directory in a distributed system is therefore low.

### 3.16.3.2 Logical file namespace

The logical namespace needs to be unique. If we really want to exploit the filesystem semantics, it would be desirable to agree on some properties of the logical file names, as it is the case also for distributed file systems. If we take AFS as an example, it declares its root to be /afs followed by the AFS cell name. A similar idea could be applied to the Grid File Catalog namespace, i.e. start with /grid followed by the virtual organization name. Below this namespace each VO can define their own structure to prevent conflicts.

### 3.16.3.3 Scalability vs. Consistency

The File Catalogs that have been deployed to date are all deployed centrally and therefore are a single point of failure. The central catalog model has obviously excellent consistency properties (concurrent writes are always managed at the same place) but it does not scale to many dozens of sites. There are two possibilities to solve this issue:

1. Database Replication. The underlying database is replicated using native database replication techniques. This however means a lock-in to a vendor-specific solution. Currently commercial database vendors like Oracle provide multi-master database replication options which could be exploited, open-source solutions are not really mature yet, so this option has its limitations.
2. Lazy database synchronization exploiting the specific semantics of the File Catalog using reliable messaging to propagate the updates. Reliable messaging technologies are available (just like database technologies under point 1) both commercially and in the open source domain. The File Catalog semantics are rather simple and very specific for catalog write operations, so that every time a local write operation occurs, it can be distributed through a reliable message queue to all remote catalogs. This way one can have the same effect as under solution 1 above but without the need for vendor lock-in.

The proposal is to go with solution 2 with solution 1 as a fall-back option. Consistency might be broken in both models i.e. it is possible to register the same LFN in two remote catalogs at

the same time such that a conflict will occur. The reconciliation techniques apply in both cases, however for case 2 we can be specific to the semantics of the system and exploit the uniqueness of the GUIDs,

#### **Additional types.**

There are additional types and structures to be precisely defined (wrt what is specified in the appendix), including simple types like timestamps but possibly complex ones as: machine data (SE), policies, tags, security objects and other types to be interchanged/reached via other services.

#### **Bulk operations**

Bulk operations have been requested by many people to increase performance and to optimise interaction with the Grid services. A simple `grid_execute` command might be the solution that takes an XML document containing all the operations that the user wants to perform in one go. The XML structure needs to be defined. Open issues involve behaviour on failures, transactional consistency, session management.

#### **Metadata, HEP CAL DataSets**

Hepcal only specifies dataset metadata, which would be. Do we also specify metadata on files?

Virtual directories are the result of a metadata query.

#### **Sessions**

Most methods in the complex and basic category can be extended to include sessions, which would add a session structure to each call to be tracked. This slows things down but of course increases the functionality and robustness. Should we aim for providing such sessioned versions of all calls?

### **3.16.4 Initial Components used for Prototype**

- AliEn File Catalog
- EDG Replica catalog (RLS)

## **3.17 METADATA CATALOG**

The Metadata Catalogue Service contains additional information (arbitrary and extensible set of attributes) about the contents of the available files. These metadata are used for querying the Metadata Catalogue in the search for the datasets meeting the required criteria.

The metadata catalog interface is defined to serve a very well defined set of tasks. The metadata can be generic or file-based. The file-based metadata always assigns metadata to logical file names or GUIDs, and has therefore more specific semantics than generic application metadata.

All metadata is application specific and therefore all metadata catalogs should optimally be provided by the applications and not the core middleware layer. There can be callouts to these catalogs from within the middleware stack through some well-defined interfaces, which the application metadata catalogs can choose to implement. The File-based metadata API is such

an interface, it's being used by the file catalog to interact with the metadata catalog. Per VO there can be only one such catalog, as there is only one file catalog.

### 3.17.1 File-based Metadata API

File-based metadata is application-specific, so only generic key-value pairs are being used at this point to define metadata.

API call	Description
grid_md_add_tag	Add a tag of a certain type to a given file
grid_md_remove_tag	Remove the tag from the file
grid_md_show_tags	Show tags and their type associated with a file.
grid_md_tag_exists	Ask whether a tag exists for a given file
grid_md_set_tag_value	Set the value of a given tag of a file
grid_md_get_tag_value	Retrieve the value of a tag of a file
grid_md_list_files_by_tag	List all files having a given value for a tag in a directory, with offset and number of results to return
grid_md_get_files_by_tag_search	Find all files matching a tag query, starting from a given directory, with offset and number of results to return.
grid_md_count_files_by_tag_search	Find the number of files matching a tag query.
grid_md_describe_tags	Describe the existing tags. This will list the schema as it is seen through the metadata API.

### 3.18 PACKAGE MANAGER

The Package Manager Service is a specialised database for the available software packages. It keeps track of the package names, versions and their locations in data repositories, usually Storage Elements. The software package dependencies information is used by installation procedures to insure coherency between the installed packages. The service provides also the information about the lifetime of the packages that is used for the clean up of the obsolete versions installed on CE's.

The packet manager relates to the data services discussed above as such that it can make use of the LFN logical namespace and of the grid storage to keep all the packets and to make them accessible to everyone through the grid. There may be a convention to keep all packets in a well-defined namespace of the LFN tree structure.

**To be added at a later phase of the project.**

## 4 REFERENCES

- [R1] D2.1 Report on Current Technology: Data Access and Mass Storage, EDG Deliverable 2.1, 20 December 2001. [http://cern.ch/edg-wp2/docs/DataGrid-02-D2.1-0105-2\\_0.pdf](http://cern.ch/edg-wp2/docs/DataGrid-02-D2.1-0105-2_0.pdf)
- [R2] Wolfgang Hoschek, Javier Jaen- Martinez, Peter Kunszt, Ben Segal, Heinz Stockinger, Kurt Stockinger, Brian Tierney, Data Management (WP2) Architecture Report , EDG Deliverable 2.2, <http://edms.cern.ch/document/332390>
- [R3] Data Management Workpackage, EU DataGrid: D2.2.A1 Addendum to the Data Management Architecture Report (Covering Testbed Release 2.0), EDG Deliverable 2.2.A1 <http://edms.cern.ch/document/374107/addendum.pdf>
- [R4] Data Management Workpackage, EU DataGrid: D2.5 Components and Documentation for the Final Project Release. <https://edms.cern.ch/file/407063/1/finalDocumentationWP2.pdf>
- [R5] DataGrid WP1, Definition of Architecture, Technical Plan and Evaluation Criteria for Scheduling, Resource Management, Security and Job Description, Technical Report, EU DataGrid Project. Deliverable D1.2, September 2001. <https://edms.cern.ch/file/332413/1/datagrid-01-d1.2-0112-0-3.pdf>
- [R6] W. Allcock, J. Bester, J. Bresnahan, A. Chernevak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnal, S. Tuecke; Data Management and Transfer in High Performance Computational Grid Environments. Parallel Computing, 2002. <http://www.globus.org/research/papers/dataMgmt.pdf>
- [R7] CASTOR web-site: <http://cern.ch/castor>
- [R8] Gregor von Laszewski, Ian Foster, Jarek Gawor, Peter Lane: A Java Commodity Grid Kit , Concurrency and Computation: Practice and Experience, 13(8-9), 2001. [http://www.globus.org/research/papers/vonLaszewski\\_cog-cpe-final.pdf](http://www.globus.org/research/papers/vonLaszewski_cog-cpe-final.pdf)
- [R9] Heinz Stockinger, Flavia Donno, Erwin Laure, Shahzad Muzaffar, Peter Kunszt, Giuseppe Andronico, and Paul Millar. Grid Data Management in Action: Experience in Running and Supporting Data Management Services in the EU DataGrid Project. In Computing in High Energy Physics (CHEP 2003), La Jolla, California, March 24-28 2003.
- [R10] Diana Bosio, James Casey, Akos Frohner, Leanne Guy, Peter Kunszt, Erwin Laure, Sophie Lemaitre, Levi Lucio, Heinz Stockinger, Kurt Stockinger, William Bell, David Cameron, Gavin Mc- Cance, Paul Millar, Joni Hahkala, Niklas Karlsson, Ville Nenonen, Mika Silander, Olle Mulmo, Gian-Luca Volpato, Giuseppe Andronico, Federico DiCarlo, Livio Salconi, Andrea Domenici, Ruben Carvajal-Schiaffino, and Floriano Zini. Next-Generation EU DataGrid Data Management Services. In Computing in High Energy Physics (CHEP 2003) , La Jolla, California, March 24-28 2003.
- [R11] Heinz Stockinger, Asad Samar, Shahzad Muzaffar, and Flavia Donno. Grid Data Mirroring Package (GDMP). Scientific Programming Journal - Special Issue: Grid Computing, 10(2):121-134, 2002.
- [R12] SRM documents. <http://sdm.lbl.gov/srm-wg/documents.html>
- [R13] H. B. Newman, I.C. Legrand, MonaLisa: A Distributed Monitoring Service Architecture, these proceedings, MOET001
- [R14] MONARC. <http://www.cern.ch/MONARC/>
- [R15] <http://doc.in2p3.fr/bbftp/>
- [R16] <http://asg.web.cmu.edu/sasl/>

- [R17] DataGrid Security Coordination Group: D7.5 Security Requirements and testbed 1 security implementation. <https://edms.cern.ch/document/340234/4.0>
- [R18] Shawn Mullen et al.: Grid Authentication Authorization and Accounting Requirements, GWD-I GGF SiteAAA RG, <http://www.ppdg.net/pa/ppdg-pa/siteaa/GGF-SiteAAA-RG/draft-ggf-saar-reqs-4.txt>

## **API Definitions**

### **API AND GRID ACCESS SERVICE**

The initial API will follow the AliEn API as depicted in Figure 11.

TAlIEnAPI  
(from C++ Reverse Engineered)

```

TAlIEnAPI()
<<virtual>> ~TAlIEnAPI()
ReuseApiServer(user : const char*) : int
StartApiServer(user : const char*) : int
StopApiServer() : int
SetApiServerUrl(url : const char*) : void
GetApiUser() : const char*
RequestPROOFSession(user : const char*, nsites : int, sites : string**, ntimes : string**, starttime : time_t, duration : time_t) : ProofSession_T*
QueryPROOFSession(proof : ProofSession_T*) : ProofSessionStatus_T
ListPROOFSessions(sessionId : int) : void
ListPROOFDaemons() : void
CancelPROOFSession(sessionId : int) : bool
CancelPROOFSession(proof : ProofSession_T*) : bool
OpenDir(cname : const char *) : RESULTHANDLE
CloseResult(hResult : const RESULTHANDLE) : void
ResetResult(hResult : const RESULTHANDLE) : void
ReadResult(hResult : const RESULTHANDLE) : const struct Result_T*
MkDir(dir : const char *, makeall : const bool = false) : int
Rmdir(dir : const char *, deleteall : const bool = false) : int
Rm(lfn : const char *, deleteall : const bool = false) : int
Cp(source : const char *, target : const char *) : int
Mv(source : const char *, target : const char *) : int
AddFile(lfn : const char *, pfn : const char *, size : int = -1, se : const char* = NULL, guid : const char* = NULL) : int
AddFileMirror(lfn : const char *, pfn : const char *, se : const char *) : int
RegisterFile(lfn : const char *, pfn : const char *) : int
GetPhysicalFileNames(lfn : const char *) : RESULTHANDLE
AddTag(lfn : const char *, tagName : const char *) : int
RemoveTag(lfn : const char *, tagName : const char *) : int
GetTags(lfn : const char *) : RESULTHANDLE
AddAttribute(lfn : const char *, tagName : const char *, attrname : const char *, attrval : const char *) : int
DeleteAttribute(lfn : const char *, tagName : const char *, attrname : const char *) : int
GetAttributes(lfn : const char *, tagName : const char *) : RESULTHANDLE
Chmod(lfn : const char*, mode : mode_t) : int
Chown(lfn : const char*, owner : const char*, group : const char*) : int
GetFile(lfn : const char*, localdest : const char* = NULL) : char*
SubmitJob(jdfile : const char *) : int
GetJobStatus(JobID : int) : JobStatus_T*
Find(path : const char*, file : const char*, conditions : const char* = NULL) : RESULTHANDLE
FindEx(path : const char*, file : const char*, conditions : const char* = NULL) : RESULTHANDLE
FindExCount(path : const char*, file : const char*, conditions : const char* = NULL) : RESULTHANDLE
KillJob(JobId : int) : int
ResubmitJob(JobId : int) : int
GetAccessPath(lfn : const char*, mode : bool = false, se : const char* = NULL) : AccessPath_T*
GetFileURL(se : const char*, path : const char*) : char*
open(lfn : const char *, mode : int, flags : mode_t) : GRIDFILEHANDLE
close(handle : GRIDFILEHANDLE) : int
read(handle : GRIDFILEHANDLE, buffer : void*, size : long, offset : long long) : int
write(handle : GRIDFILEHANDLE, buffer : void*, size : long, offset : long long) : int
fstat(handle : GRIDFILEHANDLE, statbuf : struct stat *) : int
fsync(handle : GRIDFILEHANDLE) : int
fchmod(handle : GRIDFILEHANDLE, mode : mode_t) : int
fchown(handle : GRIDFILEHANDLE, owner : uid_t, group : gid_t) : int
stat(lfn : const char*, statbuf : struct stat*) : int
lstat(lfn : const char*, statbuf : struct stat*) : int
opendir(dir : const char *) : GRIDFILEHANDLE
readdir(handle : const GRIDFILEHANDLE) : const GRIDFILEENTRY*
closedir(handle : const GRIDFILEHANDLE) : int
OpenDirInternal(cname : const char *) : RESULTHANDLE
SimpleFunc(id : const int, dir : const char *, flag : const bool) : int
SimpleFunc(func : const char *, inargs : const int, ... ) : int
GetDataInternal(id : const int, arg1 : const char *, arg2 : const char* = NULL, arg3 : const char* = NULL) : RESULTHANDLE
SoapCall(cfunction : const char*, arg : const char*, result : char**) : int
SoapCall(cfunction : const char*, arg : const char*, paramlist : TCppSoap::ParamList_T*) : int
SoapCall(cfunction : const char*, cargs : const char* [], inumarg : const int, paramlist : TCppSoap::ParamList_T*) : int
SoapCall(cfunction : const char*, cargs : const char* [], inumarg : const int, result : char**) : int
GetNextFreeResultHandle() : RESULTHANDLE
GetNextFreeGridFileHandle() : GRIDFILEHANDLE

```



Figure 11: C++ API and an TALien (implementation of ROOT TGrid ) classes



## Information and Monitoring Services

### Package rgma

#### Class Summary

##### Interfaces

<a href="#">ConsumerFactory</a>	A factory to create Consumers.
<a href="#">ProducerFactory</a>	A factory to create Primary, Secondary and On Demand Producers.
<a href="#">Resource</a>	A Resource is an object in a Web Service that the API can interact with, either a Producer or a Consumer.
<a href="#">Consumer</a>	A client uses a Consumer to retrieve data from one or more producers.
<a href="#">Producer</a>	A Producer allows a client to create, declare and undeclare tables.
<a href="#">OnDemandProducer</a>	A client uses an OnDemandProducer to publish data into R-GMA when the cost of creating each message is high.
<a href="#">PrimaryProducer</a>	A client uses a PrimaryProducer to publish information into R-GMA.
<a href="#">SecondaryProducer</a>	A client uses a secondary producer to republish or store information from other producers.

rgma

## ConsumerFactory

### Description

A factory to create Consumers.

#### Member Summary

##### Methods

Consumer	<code>createConsumer(SelectStatement select, QueryType queryType, StringList voNames)</code> Creates a consumer with a specific SELECT query and query type that will use a mediator to find relevant producers from which to receive data within the specified VO's.
----------	--

---

### Methods

#### createConsumer(SelectStatement, QueryType, StringList)

```
public rgma.Consumer createConsumer(rgma.SelectStatement select,
    rgma.QueryType queryType, rgma.StringList voNames)
    throws RemoteException, RGMAException
```

Creates a consumer with a specific SELECT query and query type that will use a mediator to find relevant producers from which to receive data. The Consumer will only consume data from the VO's listed in voNames.

##### Parameters:

select - An SQL SELECT statement.  
queryType - The type of the query.  
voNames - List of VO names to consume from.

##### Returns:

The new consumer.

##### Throws:

[RemoteException](#) - If the service could not be contacted.  
[RGMAException](#) - If the select statement is invalid.  
[RGMAException](#) - If no valid VO names are provided.  
[RGMAException](#) - If no endpoint is defined.

rgma

## ProducerFactory

### Description

A factory for Primary, Secondary and On Demand Producers.

#### Member Summary

##### Methods

OnDemandProducer	<code>createOnDemandProducer(Uri uri, ProducerType producerType, StringList voNames)</code> Creates an on-demand producer of the specified type.
PrimaryProducer	<code>createPrimaryProducer(StorageLocation storageLocation, ProducerType producerType, StringList voNames)</code> Creates a primary producer that uses the specified data storage and producer type.
SecondaryProducer	<code>createSecondaryProducer(StorageLocation storageLocation, ProducerType producerType, StringList voNames)</code> Creates a secondary producer, that uses a primary producer with the specified attributes to republish information.

---

### Methods

#### createOnDemandProducer(Uri, ProducerType, StringList)

```
public rgma.OnDemandProducer createOnDemandProducer(Uri uri,
    rgma.ProducerType producerType, rgma.StringList voNames)
    throws RemoteException, RGMAException
```

Creates an on-demand producer.

##### Parameters:

`uri` - The URI for the system that will respond to queries.

`producerType` - The type of producer.

`voNames` - List of VO names to publish to.

##### Returns:

A new on-demand producer.

##### Throws:

[RemoteException](#) - If the service could not be contacted.

[RGMAException](#) - If no endpoint is defined.

[RGMAException](#) - If the URI is invalid.

#### createPrimaryProducer(StorageLocation, ProducerType, StringList)

```
public rgma.PrimaryProducer createPrimaryProducer(rgma.StorageLocation
    storageLocation, rgma.ProducerType producerType,
```

```
    rgma.StringList voNames)
    throws RemoteException, RGMAException
```

Creates a primary producer that uses the specified data storage and producer type.

**Parameters:**

storageLocation - The location to store tuples.

producerType - The type of producer.

voNames - List of VO names to publish to.

**Returns:**

A new primary producer.

**Throws:**

[RemoteException](#) - If the service could not be contacted.

[RGMAException](#) - If no endpoint is defined.

[RGMAException](#) - If the storage location is invalid.

### **createSecondaryProducer(StorageLocation, ProducerType, StringList)**

```
public rgma.SecondaryProducer createSecondaryProducer(rgma.StorageLocation
    storageLocation, rgma.ProducerType producerType,
    rgma.StringList voNames)
    throws RemoteException, RGMAException
```

Creates a secondary producer, that uses a primary producer with the specified attributes to republish information.

**Parameters:**

storageLocation - The location to store tuples.

producerType - The type of producer.

voNames - List of VO names to publish to.

**Returns:**

A new secondary producer.

**Throws:**

[RemoteException](#) - If the service could not be contacted.

[RGMAException](#) - If no endpoint is defined.

[RGMAException](#) - If the storage location is invalid.

rgma

## Resource

### All Known Subinterfaces:

[Consumer](#), [OnDemandProducer](#), [PrimaryProducer](#), [Producer](#),  
[SecondaryProducer](#)

### Description

A Resource is an object in a Web Service with which an API can interact. In R-GMA, Producers and Consumers are both Resources.

Member Summary	
<b>Methods</b>	
void	<code>close()</code> Closes the resource.
void	<code>destroy()</code> Closes and destroys the resource.
Endpoint	<code>getEndpoint()</code> Gets the endpoint of the service/resource the API communicates with.
TimeInterval	<code>getTerminationInterval()</code> Retrieves the resource's termination interval.
void	<code>setTerminationInterval(TimeInterval terminationInterval)</code> Sets the termination interval for this resource.
void	<code>showSignOfLife()</code> Indicates to the resource that this API is still alive.

---

## Methods

### close()

```
public void close()  
    throws RemoteException, UnknownResourceException
```

Closes the resource. The resource will no longer be available through this API and will be destroyed once it has finished interacting with other components.

#### Throws:

[RemoteException](#) - If the service could not be contacted.

[UnknownResourceException](#) - If the resource could not be found

### destroy()

```
public void destroy()  
    throws RemoteException, UnknownResourceException
```

Closes and destroys the resource. The resource will no longer be available to any component.

**Throws:**

[RemoteException](#) - If the service could not be contacted.

[UnknownResourceException](#) - If the resource could not be found

**getEndpoint()**

```
public rgma.Endpoint getEndpoint()
```

Gets the endpoint of the service/resource the API communicates with.

**Returns:**

The service/instance identifier

**getTerminationInterval()**

```
public rgma.TimeInterval getTerminationInterval()  
    throws RemoteException, UnknownResourceException
```

Retrieves the resource's termination interval.

**Returns:**

The termination interval as a TimeInterval.

**Throws:**

[RemoteException](#) - If the service could not be contacted.

[UnknownResourceException](#) - If the resource could not be found

**See Also:**

[setTerminationInterval\(TimeInterval\)](#)

**setTerminationInterval(TimeInterval)**

```
public void setTerminationInterval(rgma.TimeInterval terminationInterval)  
    throws RemoteException, UnknownResourceException, RGMAException
```

Sets the termination interval for this resource.

**Parameters:**

`terminationInterval` - The time interval after which the resource may destroy itself if no contact has been made from the API.

**Throws:**

[RemoteException](#) - If the service could not be contacted.

[UnknownResourceException](#) - If the resource could not be found

[RGMAException](#) - If the termination interval is invalid (e.g. in the past)

**See Also:**

[getTerminationInterval\(\)](#)

**showSignOfLife()**

```
public void showSignOfLife()  
    throws RemoteException, UnknownResourceException
```

Indicates to the resource that this API is still alive. This should be used to keep a resource alive when no other method needs to be called.

**Throws:**

[RemoteException](#) - If the service could not be contacted.

[UnknownResourceException](#) - If the resource could not be found

rgma

## Consumer

### All Superinterfaces:

[Resource](#)

### Description

A client uses a Consumer to retrieve data from one or more producers.

#### Member Summary

##### Methods

void	<a href="#">abort()</a>	Aborts the current query.
boolean	<a href="#">canPop()</a>	Determines if there is data available to pop.
boolean	<a href="#">hasAborted()</a>	Determines if the last query has aborted.
boolean	<a href="#">isExecuting()</a>	Determines whether the last query is still executing.
ResultSet	<a href="#">pop()</a>	Retrieves all data from the consumer that it has received from producers.
ResultSet	<a href="#">pop(int maxCount)</a>	Retrieves at most maxCount tuples from the consumer that it has received from producers.
void	<a href="#">start(TimeInterval timeout)</a>	Starts this consumer's query, terminating after the specified time interval.

#### Inherited Member Summary

##### Methods inherited from interface [Resource](#)

[close\(\)](#), [destroy\(\)](#), [getEndpoint\(\)](#), [getTerminationInterval\(\)](#), [setTerminationInterval\(TimeInterval\)](#), [showSignOfLife\(\)](#)

---

### Methods

#### **abort()**

```
public void abort()
    throws UnknownResourceException, RemoteException
```

Aborts the current query.

**Throws:**

[UnknownResourceException](#) - If the consumer resource is not found.

[RemoteException](#) - If the service could not be contacted.

**canPop()**

```
public boolean canPop()  
    throws UnknownResourceException, RemoteException
```

Determines if there is data available to pop.

**Returns:**

True if the consumer's buffer contains data.

**Throws:**

[UnknownResourceException](#) - If the consumer resource is not found.

[RemoteException](#) - If the service could not be contacted.

**hasAborted()**

```
public boolean hasAborted()  
    throws UnknownResourceException, RemoteException
```

Determines if the last query has aborted.

**Returns:**

True if the query was aborted and didn't stop of its own accord.

**Throws:**

[UnknownResourceException](#) - If the consumer resource is not found.

[RemoteException](#) - If the service could not be contacted.

**isExecuting()**

```
public boolean isExecuting()  
    throws UnknownResourceException, RemoteException
```

Determines whether the last query is still executing.

**Returns:**

True if the query is still executing.

**Throws:**

[UnknownResourceException](#) - If the consumer resource is not found.

[RemoteException](#) - If the service could not be contacted.

**pop()**

```
public rgma.ResultSet pop()  
    throws UnknownResourceException, RemoteException, RGMAException
```

Retrieves all data from the consumer that it has received from producers.

**Returns:**

A ResultSet containing the received tuples.

**Throws:**

[UnknownResourceException](#) - If the consumer resource is not found.

[RemoteException](#) - If the service could not be contacted.

[RGMAException](#) - If there is no data to pop.

### **pop(int)**

```
public rgma.ResultSet pop(int maxCount)
    throws UnknownResourceException, RemoteException, RGMAException
```

Retrieves at most maxCount tuples from the consumer that it has received from producers.

#### **Parameters:**

maxCount - The maximum number of tuples to retrieve.

#### **Returns:**

A ResultSet containing the received tuples.

#### **Throws:**

[UnknownResourceException](#) - If the consumer resource is not found.

[RemoteException](#) - If the service could not be contacted.

[RGMAException](#) - If there is no data to pop.

### **start(TimeInterval)**

```
public void start(rgma.TimeInterval timeout)
    throws UnknownResourceException, RemoteException, RGMAException
```

Starts this consumer's query, terminating after the specified time interval. There is no infinite interval; just set the timeout to be very large if this is the effect you want (e.g. 1000 days).

#### **Parameters:**

timeout - Time interval after which the query will automatically be aborted.

#### **Throws:**

[UnknownResourceException](#) - If the consumer resource is not found.

[RemoteException](#) - If the service could not be contacted.

[RGMAException](#) - If the query is currently executing.

rgma

## Producer

### All Superinterfaces:

[Resource](#)

### All Known Subinterfaces:

[OnDemandProducer](#), [PrimaryProducer](#), [SecondaryProducer](#)

### Description

A Producer allows a client to create, declare and undeclare tables. A client does not use a Producer directly, but rather one of its sub-types: PrimaryProducer, SecondaryProducer or OnDemandProducer.

### See Also:

[PrimaryProducer](#), [SecondaryProducer](#), [OnDemandProducer](#)

#### Member Summary

##### Methods

```
void createTable(CreateTableStatement createTable,
                 TableAuthorization tableAuthorization)
    Creates a table, defining its structure and authorization.

void declareTable(String tableName, Predicate predicate)
    Declares a table into which this Producer can publish.

void undeclareTable(String tableName)
    Undoes a table declaration.
```

#### Inherited Member Summary

##### Methods inherited from interface [Resource](#)

```
close(), destroy(), getEndpoint(), getTerminationInterval(),
setTerminationInterval(TimeInterval), showSignOfLife()
```

---

## Methods

### **createTable(CreateTableStatement, TableAuthorization)**

```
public void createTable(rgma.CreateTableStatement createTable,  
                        rgma.TableAuthorization tableAuthorization)  
    throws RemoteException, UnknownResourceException, RGMAException
```

Creates a table, defining its structure and authorization.

#### **Parameters:**

`createTable` - An SQL CREATE TABLE statement, defining the column names, types etc for this table.

`tableAuthorization` - The authorization for this table.

#### **Throws:**

[RemoteException](#) - If the service could not be contacted.

[UnknownResourceException](#) - If the producer resource could not be found

[RGMAException](#) - If the table already exists and the `createTable` and `tableAuthorization` don't match the existing details.

[RGMAException](#) - If the CREATE TABLE statement is invalid.

[RGMAException](#) - If the table authorization is invalid.

### **declareTable(String, Predicate)**

```
public void declareTable(String tableName, rgma.Predicate predicate)  
    throws RemoteException, UnknownResourceException, RGMAException
```

Declares a table into which this Producer can publish. A subset of a table can be declared using a predicate.

#### **Parameters:**

`tableName` - The name of the table to publish into.

`predicate` - An SQL WHERE clause defining the subset of a table that this Producer will publish. To publish to the whole table, an empty predicate can be defined using `new Predicate("")` or `new Predicate()`

#### **Throws:**

[RemoteException](#) - If the service could not be contacted.

[UnknownResourceException](#) - If the producer resource could not be found

[RGMAException](#) - If the `tableName` is unknown

[RGMAException](#) - If the `createTable` does not match an existing table description

[RGMAException](#) - If the predicate is invalid

#### **See Also:**

[PrimaryProducer.declareTable\(String, Predicate, TimeInterval\)](#)

### **undeclareTable(String)**

```
public void undeclareTable(String tableName)  
    throws RemoteException, UnknownResourceException, RGMAException
```

Undoes a table declaration.

**Parameters:**

tableName - The name of the table to undeclare.

**Throws:**

[RemoteException](#) - If the service could not be contacted.

[UnknownResourceException](#) - If the producer resource could not be found

[RGMAException](#) - If the tableName has not been declared

rgma

## OnDemandProducer

### All Superinterfaces:

[Producer](#), [Resource](#)

### Description

A client uses an OnDemandProducer to publish data into R-GMA when the cost of creating each message is high. The OnDemandProducer only generates messages when there is a specific query from a Consumer.

### Inherited Member Summary

#### Methods inherited from interface [Producer](#)

```
createTable(CreateTableStatement, TableAuthorization), declareTable(String, Predicate), undeclareTable(String)
```

#### Methods inherited from interface [Resource](#)

```
close(), destroy(), getEndpoint(), getTerminationInterval(), setTerminationInterval(TimeInterval), showSignOfLife()
```

rgma

## PrimaryProducer

### All Superinterfaces:

[Producer](#), [Resource](#)

### Description

A client uses a PrimaryProducer to publish information into R-GMA.

#### Member Summary

##### Methods

```
void declareTable(String tableName, Predicate predicate,
                  TimeInterval minRetentionPeriod)
    Declares a table, specifying the minimum time period for which the tuple must be
    retained.

void insert(InsertStatement insert)
    Publishes data by inserting a tuple into a table, both specified by the SQL INSERT
    statement.

void insert(InsertStatementList inserts)
    Publishes using a list of SQL INSERT statements.
```

#### Inherited Member Summary

##### Methods inherited from interface [Producer](#)

```
createTable(CreateTableStatement, TableAuthorization), declareTable(String,
Predicate), undeclareTable(String)
```

##### Methods inherited from interface [Resource](#)

```
close(), destroy(), getEndpoint(), getTerminationInterval(),
setTerminationInterval(TimeInterval), showSignOfLife()
```

---

### Methods

#### declareTable(String, Predicate, TimeInterval)

```
public void declareTable(String tableName, rgma.Predicate predicate,
                        rgma.TimeInterval minRetentionPeriod)
    throws RemoteException, UnknownResourceException, RGMAException
```

Declares a table, specifying the minimum time period for which the tuple must be retained. Beyond this, a tuple will be removed when there are no longer any consumers interested in it.

**Parameters:**

- `tableName` - The name of the table to declare.
- `predicate` - An SQL WHERE clause defining the subset of a table that this Producer will publish. To publish to the whole table, an empty predicate can be used.
- `minRetentionPeriod` - The minimum time for which tuples must be retained.

**Throws:**

- [RemoteException](#) - If the service could not be contacted.
- [UnknownResourceException](#) - If the producer resource could not be found
- [RGMAException](#) - If the `tableName` is unknown
- [RGMAException](#) - If the predicate is invalid
- [RGMAException](#) - If the `minRetentionPeriod` is invalid

**See Also:**

[Producer.declareTable\(String, Predicate\)](#)

**insert(InsertStatement)**

```
public void insert(rgma.InsertStatement insert)
    throws RemoteException, UnknownResourceException, RGMAException
```

Publishes data by inserting a tuple into a table, both specified by the SQL INSERT statement.

**Parameters:**

- `insert` - An SQL INSERT statement providing the data to publish and the table into which to put it.

**Throws:**

- [RemoteException](#) - If the service could not be contacted.
- [UnknownResourceException](#) - If the producer resource could not be found
- [RGMAException](#) - If the table in the insert has not been declared
- [RGMAException](#) - If the inserted tuple does not match its table's declared predicate

**insert(InsertStatementList)**

```
public void insert(rgma.InsertStatementList inserts)
    throws RemoteException, UnknownResourceException, RGMAException
```

Publishes using a list of SQL INSERT statements.

**Parameters:**

- `inserts` - A list of SQL INSERT statements.

**Throws:**

- [RemoteException](#) - If the service could not be contacted.
- [UnknownResourceException](#) - If the producer resource could not be found
- [RGMAException](#) - If the table in any insert has not been declared
- [RGMAException](#) - If any inserted tuple does not match its table's declared predicate

**See Also:**

[insert\(InsertStatement\)](#)



rgma

## SecondaryProducer

### All Superinterfaces:

[Producer](#), [Resource](#)

### Description

A client uses a secondary producer to republish or store information from other producers.

#### Inherited Member Summary

##### Methods inherited from interface [Producer](#)

```
createTable(CreateTableStatement, TableAuthorization), declareTable(String, Predicate), undeclareTable(String)
```

##### Methods inherited from interface [Resource](#)

```
close(), destroy(), getEndpoint(), getTerminationInterval(), setTerminationInterval(TimeInterval), showSignOfLife()
```

#### Classes

<a href="#">CreateTableStatement</a>	An SQL CREATE TABLE statement.
<a href="#">Endpoint</a>	A service or resource endpoint.
<a href="#">EndpointList</a>	A list of Endpoint objects.
<a href="#">InsertStatement</a>	An SQL INSERT statement.
<a href="#">InsertStatementList</a>	A list of InsertStatement objects.
<a href="#">Predicate</a>	An SQL WHERE clause.
<a href="#">ProducerType</a>	The type of a producer, specifically the data set that it will store.
<a href="#">QueryType</a>	The type of a consumer query.
<a href="#">ResultSet</a>	A set of tuples, modelled on the java.sql.ResultSet and providing a subset of its functionality.
<a href="#">ResultSetMetaData</a>	Column and table details for a ResultSet, modelled on java.sql.ResultSetMetaData.

<a href="#">SelectStatement</a>	An SQL SELECT statement.
<a href="#">SQLStatement</a>	A generic SQL statement.
<a href="#">StorageLocation</a>	The location where tuples will be stored.
<a href="#">StringList</a>	A list of String objects.
<a href="#">TableAuthorization</a>	Authorization rules for a table.
<a href="#">TimeInterval</a>	Encapsulates a time value and the units being used.
<a href="#">Types</a>	Constants for SQL column types.
<a href="#">Units</a>	Time units.
<b>Exceptions</b>	
<a href="#">RemoteException</a>	An exception thrown when the API is unable to contact a service (e.g.
<a href="#">RGMAException</a>	Exception thrown when an error occurs in the RGMA application.
<a href="#">UnknownResourceException</a>	Exception thrown when a resource cannot be found in the resource framework.

## CreateTableStatement

### Declaration

```
public class CreateTableStatement extends SQLStatement
```

```
rgma.SQLStatement
|
+--rgma.CreateTableStatement
```

### Description

An SQL CREATE TABLE statement.

#### Member Summary

##### Constructors

```
CreateTableStatement(String createTableStatement)
Creates a new CreateTableStatement from a String of SQL.
```

## Inherited Member Summary

Methods inherited from class [SQLStatement](#)

[getStatement\(\)](#)

---

## Constructors

### CreateTableStatement(String)

```
public CreateTableStatement(String createTableStatement)
```

Creates a new CreateTableStatement from a String of SQL.

**Parameters:**

createTableStatement - An SQL CREATE TABLE statement as a String.

rgma

## Endpoint

### Declaration

```
public class Endpoint
```

```
rgma.Endpoint
```

### Description

A service or resource endpoint.

### Member Summary

#### Constructors

```
Endpoint(String url)
```

Creates a new Endpoint object.

```
Endpoint(String url, int resourceId)
```

Creates a new Endpoint object.

#### Methods

```
int getResourceId()
```

Gets the resource identifier.

```
String getUrl()
```

Gets the service URL.

---

### Constructors

#### Endpoint(String)

```
public Endpoint(String url)
```

Creates a new Endpoint object.

##### Parameters:

url - The service URL.

#### Endpoint(String, int)

```
public Endpoint(String url, int resourceId)
```

Creates a new Endpoint object.

##### Parameters:

url - The service URL.

resourceId - The resource identified.

---

## Methods

### **getResourceId()**

```
public int getResourceId()
```

Gets the resource identifier.

**Returns:**

The resourceId.

### **getUrl()**

```
public String getUrl()
```

Gets the service URL.

**Returns:**

The service URL

rgma

## EndpointList

### Declaration

```
public class EndpointList
```

```
rgma.EndpointList
```

### Description

A list of Endpoint objects.

#### Member Summary

##### Constructors

```
EndpointList()  
Creates a new EndpointList object.
```

##### Methods

```
void addEndpoint(Endpoint endpoint)  
Add a Endpoint.  
void addEndpoint(int index, Endpoint endpoint)  
Add a Endpoint at the specified index.  
Endpoint getEndpoint(int index)  
Retrieves the Endpoint with the specified index.  
int size()  
Returns the size of the list.
```

---

### Constructors

#### EndpointList()

```
public EndpointList()  
Creates a new EndpointList object.
```

---

### Methods

#### addEndpoint(Endpoint)

```
public void addEndpoint(rgma.Endpoint endpoint)  
Add a Endpoint.
```

##### Parameters:

endpoint - Endpoint to add.

**addEndpoint(int, Endpoint)**

```
public void addEndpoint(int index, rgma.Endpoint endpoint)
```

Add a Endpoint at the specified index.

**Parameters:**

index - Index in list.

endpoint - Endpoint to add.

**getEndpoint(int)**

```
public rgma.Endpoint getEndpoint(int index)
```

Retrieves the Endpoint with the specified index.

**Parameters:**

index - Index of Endpoint to get (  $\leq$  index  $<$  size)

**Returns:**

A Endpoint object.

**size()**

```
public int size()
```

Returns the size of the list.

**Returns:**

The number of Endpoints in the list.

rgma

## InsertStatement

### Declaration

```
public class InsertStatement extends SQLStatement
```

```
rgma.SQLStatement  
|  
+--rgma.InsertStatement
```

### Description

An SQL INSERT statement.

#### Member Summary

##### Constructors

```
InsertStatement\(String insertStatement\)  
Creates a new InsertStatement from a String of SQL.
```

#### Inherited Member Summary

##### Methods inherited from class [SQLStatement](#)

```
getStatement\(\)
```

---

## Constructors

### InsertStatement(String)

```
public InsertStatement(String insertStatement)
```

Creates a new InsertStatement from a String of SQL.

#### Parameters:

`insertStatement` - An SQL INSERT statement as a String.

rgma

## InsertStatementList

### Declaration

```
public class InsertStatementList  
  
rgma.InsertStatementList
```

### Description

A list of InsertStatement objects.

Member Summary	
<b>Constructors</b>	
	<pre>InsertStatementList() Creates a new InsertStatementList object.</pre>
<b>Methods</b>	
	<pre>void addInsertStatement(InsertStatement insertstatement) Add a InsertStatement.</pre>
	<pre>void addInsertStatement(int index, InsertStatement insertstatement) Add a InsertStatement at the specified index.</pre>
InsertStatement	<pre>getInsertStatement(int index) Retrieves the InsertStatement with the specified index.</pre>
	<pre>int size() Returns the size of the list.</pre>

---

### Constructors

#### InsertStatementList()

```
public InsertStatementList()  
Creates a new InsertStatementList object.
```

---

### Methods

#### addInsertStatement(InsertStatement)

```
public void addInsertStatement(rgma.InsertStatement insertstatement)  
Add a InsertStatement.
```

**Parameters:**

insertstatement - InsertStatement to add.

**addInsertStatement(int, InsertStatement)**

```
public void addInsertStatement(int index,  
    rgma.InsertStatement insertstatement)
```

Add a InsertStatement at the specified index.

**Parameters:**

index - Index in list.

insertstatement - InsertStatement to add.

**getInsertStatement(int)**

```
public rgma.InsertStatement getInsertStatement(int index)
```

Retrieves the InsertStatement with the specified index.

**Parameters:**

index - Index of InsertStatement to get (  $\leq$  index  $<$  size)

**Returns:**

An InsertStatement object.

**size()**

```
public int size()
```

Returns the size of the list.

**Returns:**

The number of InsertStatements in the list.

rgma

## ProducerType

### Declaration

```
public class ProducerType
```

```
rgma.ProducerType
```

### Description

The type of a producer, specifically the data set that it will store. Either LATEST for just the most recent tuples for each primary key, or HISTORY for all tuples.

### Member Summary

#### Fields

```
static ProducerType HISTORY  
    Store all the tuples.
```

```
static ProducerType LATEST  
    Store just the latest tuples.
```

#### Methods

```
boolean equals(ProducerType pt)  
    Compares two producer type objects.
```

```
boolean isHistory()  
    Determines if this is a HISTORY producer.
```

```
boolean isLatest()  
    Determines if this is a LATEST producer.
```

---

### Fields

#### HISTORY

```
public static final rgma.ProducerType HISTORY  
    Store all the tuples.
```

#### LATEST

```
public static final rgma.ProducerType LATEST  
    Store just the latest tuples.
```

---

## Methods

### **equals(ProducerType)**

```
public boolean equals(rgma.ProducerType pt)
```

Compares two producer type objects.

#### **Parameters:**

pt - Another ProducerType

#### **Returns:**

True if the ProducerTypes are equal

### **isHistory()**

```
public boolean isHistory()
```

Determines if this is a HISTORY producer.

#### **Returns:**

true if a HISTORY producer, otherwise false.

### **isLatest()**

```
public boolean isLatest()
```

Determines if this is a LATEST producer.

#### **Returns:**

true if a LATEST producer, otherwise false.

rgma

## QueryType

### Declaration

```
public class QueryType
```

```
rgma.QueryType
```

### Description

The type of a consumer query. There are four types:

- HISTORY: `QueryType.HISTORY`
- LATEST: `QueryType.LATEST`
- CONTINUOUS: `QueryType.CONTINUOUS`
- CONTINUOUS over a time interval: `new QueryType(TimeInterval)`

### Member Summary

#### Fields

```
static QueryType CONTINUOUS
    Stream data continuously, starting with the next inserted tuple.

static QueryType HISTORY
    Retrieve data from HISTORY producers.

static QueryType LATEST
    Retrieve data from LATEST producers.
```

#### Constructors

```
QueryType(TimeInterval timeInterval)
    Creates a new CONTINUOUS QueryType object with a time interval.
```

#### Methods

```
boolean equals(QueryType qt)
    Compares two QueryTypes.

boolean hasTimeInterval()
    Determines if this QueryType has an associated TimeInterval.

boolean isContinuous()
    Determines if this is a CONTINUOUS query.

boolean isHistory()
    Determines if this is a HISTORY query.

boolean isLatest()
    Determines if this is a LATEST query.
```

---

## Fields

### CONTINUOUS

```
public static final rgma.QueryType CONTINUOUS
```

Stream data continuously, starting with the next inserted tuple.

### HISTORY

```
public static final rgma.QueryType HISTORY
```

Retrieve data from HISTORY producers.

### LATEST

```
public static final rgma.QueryType LATEST
```

Retrieve data from LATEST producers.

---

## Constructors

### QueryType(TimeInterval)

```
public QueryType(rgma.TimeInterval timeInterval)
```

Creates a new CONTINUOUS QueryType object with a time interval.

**Parameters:**

`timeInterval` - The period over which to retrieve data.

---

## Methods

### equals(QueryType)

```
public boolean equals(rgma.QueryType qt)
```

Compares two QueryTypes.

**Parameters:**

`qt` - Another QueryType.

**Returns:**

True if the QueryTypes are equal.

### hasTimeInterval()

```
public boolean hasTimeInterval()
```

Determines if this QueryType has an associated TimeInterval.

**Returns:**

true if it has a TimeInterval, otherwise false.

### isContinuous()

```
public boolean isContinuous()
```

Determines if this is a CONTINUOUS query.

**Returns:**

true if a CONTINUOUS query, otherwise false.

**isHistory()**

```
public boolean isHistory()
```

Determines if this is a HISTORY query.

**Returns:**

true if a HISTORY query, otherwise false.

**isLatest()**

```
public boolean isLatest()
```

Determines if this is a LATEST query.

**Returns:**

true if a LATEST query, otherwise false.

rgma

## RemoteException

### Declaration

```
public class RemoteException
```

```
rgma.RemoteException
```

### Description

An exception thrown when the API is unable to contact a service (e.g. Producer, ConsumerFactory).

### Member Summary

#### Constructors

```
RemoteException()  
Creates a new RemoteException object.  
  
RemoteException(String message)  
Creates a new RemoteException object.
```

---

### Constructors

#### RemoteException()

```
public RemoteException()  
Creates a new RemoteException object.
```

#### RemoteException(String)

```
public RemoteException(String message)  
Creates a new RemoteException object.
```

#### Parameters:

message - Error message

rgma

## ResultSet

### Declaration

```
public class ResultSet
```

```
rgma.ResultSet
```

### Description

A set of tuples, modelled on the java.sql.ResultSet and providing a subset of its functionality.

### Member Summary

#### Methods

```
void afterLast\(\)  
    Moves the cursor to the end of this ResultSet object, just after the last row.  
void beforeFirst\(\)  
    Moves the cursor to the front of this ResultSet object, just before the first row.  
int findColumn\(String columnName\)  
    Maps the given ResultSet column name to its ResultSet column index.  
boolean first\(\)  
    Moves the cursor to the first row in this ResultSet object.  
int getInt\(int columnIndex\)  
    Retrieves the value of the designated column in the current row of this ResultSet  
    object as an int in the Java programming language.  
int getInt\(String columnName\)  
    Retrieves the value of the designated column in the current row of this ResultSet  
    object as an int in the Java programming language.  
ResultSetMetaData getMetaData\(\)  
    Retrieves the number, types and properties of this ResultSet object's columns.  
double getReal\(int columnIndex\)  
    Retrieves the value of the designated column in the current row of this ResultSet  
    object as a double in the Java programming language.  
double getReal\(String columnName\)  
    Retrieves the value of the designated column in the current row of this ResultSet  
    object as a double in the Java programming language.  
int getRow\(\)  
    Retrieves the current row number.  
String getString\(int columnIndex\)  
    Retrieves the value of the designated column in the current row of this ResultSet  
    object as a String in the Java programming language.  
String getString\(String columnName\)  
    Retrieves the value of the designated column in the current row of this ResultSet  
    object as a String in the Java programming language.  
boolean isAfterLast\(\)  
    Retrieves whether the cursor is after the last row in this ResultSet object.  
boolean isBeforeFirst\(\)  
    Retrieves whether the cursor is before the first row in this ResultSet object.  
boolean isFirst\(\)  
    Retrieves whether the cursor is on the first row of this ResultSet object.
```

```
boolean isLast()
    Retrieves whether the cursor is on the last row of this ResultSet object.

boolean last()
    Moves the cursor to the last row in this ResultSet object.

boolean next()
    Moves the cursor down one row from its current position.

boolean previous()
    Moves the cursor to the previous row in this ResultSet object.

boolean wasNull()
    Reports whether the last column read had a value of SQL NULL.
```

---

## Methods

### **afterLast()**

```
public void afterLast()
    throws RGMAException
```

Moves the cursor to the end of this `ResultSet` object, just after the last row. This method has no effect if the result set contains no rows.

**Throws:**

`RGMAException` - if an RGMA access error occurs or the result set type is `isTYPE_FORWARD_ONLY`

### **beforeFirst()**

```
public void beforeFirst()
    throws RGMAException
```

Moves the cursor to the front of this `ResultSet` object, just before the first row. This method has no effect if the result set contains no rows.

**Throws:**

`RGMAException` - if an RGMA access error occurs or the result set type is `isTYPE_FORWARD_ONLY`

### **findColumn(String)**

```
public int findColumn(String columnName)
    throws RGMAException
```

Maps the given `ResultSet` column name to its `ResultSet` column index.

**Parameters:**

`columnName` - the name of the column

**Returns:**

the column index of the given column name

**Throws:**

`RGMAException` - if the `ResultSet` object does not contain `columnName` or an RGMA access error occurs

### **first()**

```
public boolean first()  
    throws RGMAException
```

Moves the cursor to the first row in this `ResultSet` object.

**Returns:**

true if the cursor is on a valid row; false if there are no rows in the result set

**Throws:**

[RGMAException](#) - if an RGMA access error occurs or the result set type is `TYPE_FORWARD_ONLY`

### **getInt(int)**

```
public int getInt(int columnIndex)  
    throws RGMAException
```

Retrieves the value of the designated column in the current row of this `ResultSet` object as an `int` in the Java programming language.

**Parameters:**

`columnIndex` - the first column is 1, the second is 2, ...

**Returns:**

the column value; if the value is SQL NULL, the value returned is

**Throws:**

[RGMAException](#) - if an RGMA access error occurs

### **getInt(String)**

```
public int getInt(String columnName)  
    throws RGMAException
```

Retrieves the value of the designated column in the current row of this `ResultSet` object as an `int` in the Java programming language.

**Parameters:**

`columnName` - the SQL name of the column

**Returns:**

the column value; if the value is SQL NULL, the value returned is

**Throws:**

[RGMAException](#) - if an RGMA access error occurs

### **getMetaData()**

```
public rgma.ResultSetMetaData getMetaData()  
    throws RGMAException
```

Retrieves the number, types and properties of this `ResultSet` object's columns.

**Returns:**

the description of this `ResultSet` object's columns

**Throws:**

[RGMAException](#) - if an RGMA access error occurs

### **getReal(int)**

```
public double getReal(int columnIndex)
    throws RGMAException
```

Retrieves the value of the designated column in the current row of this `ResultSet` object as a `double` in the Java programming language.

**Parameters:**

`columnIndex` - the first column is 1, the second is 2, ...

**Returns:**

the column value; if the value is SQL `NULL`, the value returned is

**Throws:**

[RGMAException](#) - if an RGMA access error occurs

### **getReal(String)**

```
public double getReal(String columnName)
    throws RGMAException
```

Retrieves the value of the designated column in the current row of this `ResultSet` object as a `double` in the Java programming language.

**Parameters:**

`columnName` - the SQL name of the column

**Returns:**

the column value; if the value is SQL `NULL`, the value returned is

**Throws:**

[RGMAException](#) - if an RGMA access error occurs

### **getRow()**

```
public int getRow()
    throws RGMAException
```

Retrieves the current row number. The first row is number 1, the second number 2, and so on.

**Returns:**

the current row number; if there is no current row

**Throws:**

[RGMAException](#) - if an RGMA access error occurs

### **getString(int)**

```
public String getString(int columnIndex)
    throws RGMAException
```

Retrieves the value of the designated column in the current row of this `ResultSet` object as a `String` in the Java programming language.

**Parameters:**

`columnIndex` - the first column is 1, the second is 2, ...

**Returns:**

the column value; if the value is SQL `NULL`, the value returned is `null`

**Throws:**

[RGMAException](#) - if an RGMA access error occurs

**getString(String)**

```
public String getString(String columnName)
    throws RGMAException
```

Retrieves the value of the designated column in the current row of this `ResultSet` object as a `String` in the Java programming language.

**Parameters:**

columnName - the SQL name of the column

**Returns:**

the column value; if the value is SQL NULL, the value returned is null

**Throws:**

[RGMAException](#) - if an RGMA access error occurs

**isAfterLast()**

```
public boolean isAfterLast()
    throws RGMAException
```

Retrieves whether the cursor is after the last row in this `ResultSet` object.

**Returns:**

true if the cursor is after the last row; false if the cursor is at any other position or the result set contains no rows

**Throws:**

[RGMAException](#) - if an RGMA access error occurs

**isBeforeFirst()**

```
public boolean isBeforeFirst()
    throws RGMAException
```

Retrieves whether the cursor is before the first row in this `ResultSet` object.

**Returns:**

true if the cursor is before the first row; false if the cursor is at any other position or the result set contains no rows

**Throws:**

[RGMAException](#) - if an RGMA access error occurs

**isFirst()**

```
public boolean isFirst()
    throws RGMAException
```

Retrieves whether the cursor is on the first row of this `ResultSet` object.

**Returns:**

true if the cursor is on the first row; false otherwise

**Throws:**

[RGMAException](#) - if an RGMA access error occurs

### **isLast()**

```
public boolean isLast()  
    throws RGMAException
```

Retrieves whether the cursor is on the last row of this `ResultSet` object. Note: Calling the method `isLast` may be expensive because the JDBC driver might need to fetch ahead one row in order to determine whether the current row is the last row in the result set.

**Returns:**

`true` if the cursor is on the last row; `false` otherwise

**Throws:**

[RGMAException](#) - if an RGMA access error occurs

### **last()**

```
public boolean last()  
    throws RGMAException
```

Moves the cursor to the last row in this `ResultSet` object.

**Returns:**

`true` if the cursor is on a valid row; `false` if there are no rows in the result set

**Throws:**

[RGMAException](#) - if an RGMA access error occurs or the result set type is `isTYPE_FORWARD_ONLY`

### **next()**

```
public boolean next()  
    throws RGMAException
```

Moves the cursor down one row from its current position. A `ResultSet` cursor is initially positioned before the first row; the first call to the method `next` makes the first row the current row; the second call makes the second row the current row, and so on.

If an input stream is open for the current row, a call to the method `next` will implicitly close it. A `ResultSet` object's warning chain is cleared when a new row is read.

**Returns:**

`true` if the new current row is valid; `false` if there are no more rows

**Throws:**

[RGMAException](#) - if an RGMA access error occurs

### **previous()**

```
public boolean previous()  
    throws RGMAException
```

Moves the cursor to the previous row in this `ResultSet` object.

**Returns:**

`true` if the cursor is on a valid row; `false` if it is off the result set

**Throws:**

[RGMAException](#) - if an RGMA access error occurs or the result set type is `isTYPE_FORWARD_ONLY`

**wasNull()**

```
public boolean wasNull()  
    throws RGMAException
```

Reports whether the last column read had a value of SQL NULL. Note that you must first call one of the getter methods on a column to try to read its value and then call the method `wasNull` to see if the value read was SQL NULL.

**Returns:**

`true` if the last column value read was SQL NULL and `false` otherwise

**Throws:**

[RGMAException](#) - if an RGMA access error occurs

rgma

## ResultSetMetaData

### Declaration

```
public class ResultSetMetaData
```

```
rgma.ResultSetMetaData
```

### Description

Column and table details for a ResultSet, modelled on java.sql.ResultSetMetaData.

#### Member Summary

##### Constructors

```
ResultSetMetaData\(\)
```

##### Methods

```
int getColumnCount\(\)  
Returns the number of columns in this ResultSet object.
```

```
String getColumnName\(int column\)  
Get the designated column's name.
```

```
int getColumnType\(int column\)  
Retrieves the designated column's SQL type.
```

```
String getColumnTypeName\(int column\)  
Retrieves the designated column's database-specific type name.
```

```
String getTableName\(int column\)  
Gets the designated column's table name.
```

---

## Constructors

### ResultSetMetaData()

```
public ResultSetMetaData()
```

---

## Methods

### getColumnCount()

```
public int getColumnCount()  
throws RGMAException
```

Returns the number of columns in this ResultSet object.

#### Returns:

the number of columns

**Throws:**

[RGMAException](#) - if an RGMA access error occurs

**getColumnName(int)**

```
public String getColumnName(int column)
    throws RGMAException
```

Get the designated column's name.

**Parameters:**

column - the first column is 1, the second is 2, ...

**Returns:**

column name

**Throws:**

[RGMAException](#) - if an RGMA access error occurs

**getColumnType(int)**

```
public int getColumnType(int column)
    throws RGMAException
```

Retrieves the designated column's SQL type.

**Parameters:**

column - the first column is 1, the second is 2, ...

**Returns:**

SQL type from `rgma.Types`

**Throws:**

[RGMAException](#) - if an RGMA access error occurs

**See Also:**

[Types](#)

**getColumnTypeName(int)**

```
public String getColumnTypeName(int column)
    throws RGMAException
```

Retrieves the designated column's database-specific type name.

**Parameters:**

column - the first column is 1, the second is 2, ...

**Returns:**

type name used by the database. If the column type is a user-defined type, then a fully-qualified type name is returned.

**Throws:**

[RGMAException](#) - if an RGMA access error occurs

**getTableName(int)**

```
public String getTableName(int column)
    throws RGMAException
```

Gets the designated column's table name.

**Parameters:**

column - the first column is 1, the second is 2, ...

**Returns:**

table name or "" if not applicable

**Throws:**

[RGMAException](#) - if an RGMA access error occurs

rgma

## RGMAException

### Declaration

```
public class RGMAException
```

```
rgma.RGMAException
```

### Description

Exception thrown when an error occurs in the RGMA application.

Member Summary	
<b>Constructors</b>	
	<code>RGMAException(int errNo)</code> Creates a new RGMAException object.
	<code>RGMAException(int errNo, String message)</code> Creates a new RGMAException object.
<b>Methods</b>	
	<code>int getErrorNumber()</code> Gets the error number associated with this exception.

---

### Constructors

#### RGMAException(int)

```
public RGMAException(int errNo)
```

Creates a new RGMAException object.

**Parameters:**

`errNo` - Error number

#### RGMAException(int, String)

```
public RGMAException(int errNo, String message)
```

Creates a new RGMAException object.

**Parameters:**

`message` - Error message

`errNo` - Error number

---

## Methods

### **getErrorNumber()**

```
public int getErrorNumber()
```

Gets the error number associated with this exception.

#### **Returns:**

An error number.

rgma

## SelectStatement

### Declaration

```
public class SelectStatement extends SQLStatement
```

```
rgma.SQLStatement  
|  
+--rgma.SelectStatement
```

### Description

An SQL SELECT statement.

#### Member Summary

##### Constructors

```
SelectStatement\(String selectStatement\)  
Constructs a SelectStatement from an SQL SELECT statement string.
```

#### Inherited Member Summary

##### Methods inherited from class [SQLStatement](#)

```
getStatement\(\)
```

---

## Constructors

### SelectStatement(String)

```
public SelectStatement(String selectStatement)
```

Constructs a SelectStatement from an SQL SELECT statement string.

#### Parameters:

`selectStatement` - An SQL SELECT statement.

rgma

## SQLStatement

### Declaration

```
public class SQLStatement
```

```
rgma.SQLStatement
```

### Direct Known Subclasses:

[CreateTableStatement](#), [InsertStatement](#), [Predicate](#), [SelectStatement](#)

### Description

A generic SQL statement.

Member Summary	
<b>Constructors</b>	
	<code>SQLStatement(String statement)</code> Creates a new SQLStatement object.
<b>Methods</b>	
	<code>String getStatement()</code> Gets the SQL statement as a String.

---

## Constructors

### SQLStatement(String)

```
public SQLStatement(String statement)
```

Creates a new SQLStatement object.

#### Parameters:

statement - An SQL statement

---

## Methods

### getStatement()

```
public String getStatement()
```

Gets the SQL statement as a String.

#### Returns:

The SQL statement as a String.

rgma

## StorageLocation

### Declaration

```
public class StorageLocation
```

```
rgma.StorageLocation
```

### Description

The location where tuples will be stored. There are three options:

- MEMORY: `StorageLocation.MEMORY`
- default DATABASE: `StorageLocation.DATABASE`
- specific DATABASE: `new StorageLocation(url, user, password)`

### Member Summary

#### Fields

```
static StorageLocation DATABASE
    The default DATABASE storage location.

static StorageLocation MEMORY
    The default MEMORY storage location.
```

#### Constructors

```
StorageLocation(String url, String userName, String password)
    Creates a new DATABASE StorageLocation object.
```

#### Methods

```
boolean equals(StorageLocation p_storageLocation)
    Compares this StorageLocation to the specified location.

String getPassword()
    Gets the password for this storage system.

String getUrl()
    Gets the URL for this storage system.

String getUsername()
    Gets the user name for this storage system.

boolean hasDetails()
    Determines if this StorageLocation contains storage details.

boolean isDatabase()
    Determines the type of storage system.

boolean isMemory()
    Determines the type of storage system.
```

---

## Fields

### DATABASE

```
public static final rgma.StorageLocation DATABASE
```

The default DATABASE storage location.

### MEMORY

```
public static final rgma.StorageLocation MEMORY
```

The default MEMORY storage location.

---

## Constructors

### StorageLocation(String, String, String)

```
public StorageLocation(String url, String userName, String password)
```

Creates a new DATABASE StorageLocation object.

#### Parameters:

- `url` - The URL of the storage system (e.g. dbms)
- `userName` - The user name for the storage system.
- `password` - The password for the storage system.

---

## Methods

### equals(StorageLocation)

```
public boolean equals(rgma.StorageLocation p_storageLocation)
```

Compares this StorageLocation to the specified location.

#### Parameters:

- `p_storageLocation` - Another StorageLocation object.

#### Returns:

- true if the specified location is identical to this location.

### getPassword()

```
public String getPassword()
```

Gets the password for this storage system.

#### Returns:

- Password as a String

### getUrl()

```
public String getUrl()
```

Gets the URL for this storage system.

#### Returns:

URL as a String

### **getUserName()**

```
public String getUserName()
```

Gets the user name for this storage system.

#### **Returns:**

User name as a String

### **hasDetails()**

```
public boolean hasDetails()
```

Determines if this StorageLocation contains storage details.

#### **Returns:**

true if it has storage details, otherwise false.

### **isDatabase()**

```
public boolean isDatabase()
```

Determines the type of storage system.

#### **Returns:**

True if the storage system is a database

### **isMemory()**

```
public boolean isMemory()
```

Determines the type of storage system.

#### **Returns:**

True if the storage system is memory

rgma

## StringList

### Declaration

```
public class StringList  
  
rgma.StringList
```

### Description

A list of String objects.

#### Member Summary

##### Constructors

```
StringList()  
Creates a new StringList object.
```

##### Methods

```
void addString(int index, String string)  
Add a String at the specified index.  
  
void addString(String string)  
Add a String.  
  
String getString(int index)  
Retrieves the String with the specified index.  
  
int size()  
Returns the size of the list.
```

---

## Constructors

### StringList()

```
public StringList()  
Creates a new StringList object.
```

---

## Methods

### addString(int, String)

```
public void addString(int index, String string)  
Add a String at the specified index.
```

#### Parameters:

index - Index in list.

string - String to add.

### **addString(String)**

```
public void addString(String string)
```

Add a String.

#### **Parameters:**

string - String to add.

### **getString(int)**

```
public String getString(int index)
```

Retrieves the String with the specified index.

#### **Parameters:**

index - Index of String to get (  $\leq$  index  $<$  size)

#### **Returns:**

A String object.

### **size()**

```
public int size()
```

Returns the size of the list.

#### **Returns:**

The number of Strings in the list.

rgma

## TableAuthorization

### Declaration

```
public class TableAuthorization
```

```
rgma.TableAuthorization
```

### Description

Authorization rules for a table.

#### Member Summary

##### Constructors

```
TableAuthorization()
```

##### Methods

```
void addRule(String rule)  
    Adds a rule to this TableAuthorization.
```

```
int getNumRules()  
    Gets the number of rules in this TableAuthorization.
```

```
String getRule(int ruleNum)  
    Gets the specified rule.
```

---

## Constructors

### TableAuthorization()

```
public TableAuthorization()
```

---

## Methods

### addRule(String)

```
public void addRule(String rule)
```

Adds a rule to this TableAuthorization. A rule consists of a view on a table (a parameterized SELECT statement) and a set of allowed credentials specifying which users can access the view.

#### Parameters:

`rule` - An authorization rule.

**getNumRules()**

```
public int getNumRules()
```

Gets the number of rules in this TableAuthorization.

**Returns:**

The number of rules.

**getRule(int)**

```
public String getRule(int ruleNum)
```

Gets the specified rule.

**Parameters:**

ruleNum - Number of the rule to retrieve.

**Returns:**

The specified rule.

rgma

## TimeInterval

### Declaration

```
public class TimeInterval
```

```
rgma.TimeInterval
```

### Description

Encapsulates a time value and the units being used.

#### Member Summary

##### Constructors

```
TimeInterval(long value, Units units)  
Creates a new TimeInterval object.
```

##### Methods

```
long getValueAs(Units units)  
Gets the length of the time interval in the specified units.
```

---

## Constructors

### TimeInterval(long, Units)

```
public TimeInterval(long value, rgma.Units units)
```

Creates a new TimeInterval object.

#### Parameters:

`value` - The number of time units

`units` - The time units: MS, SECONDS, MINUTES, HOURS, DAYS

---

## Methods

### getValueAs(Units)

```
public long getValueAs(rgma.Units units)
```

Gets the length of the time interval in the specified units.

#### Parameters:

`units` - The time units: MS, SECONDS, MINUTES, HOURS, DAYS

#### Returns:

The time interval in the given units.

rgma

## Types

### Declaration

```
public class Types
```

```
rgma.Types
```

### Description

Constants for SQL column types.

#### Member Summary

##### Fields

<code>static int</code>	<code>CHAR</code>	The constant in the Java programming language, sometimes referred to as a type code, that identifies the generic SQL type CHAR.
<code>static int</code>	<code>DATE</code>	The constant in the Java programming language, sometimes referred to as a type code, that identifies the generic SQL type DATE.
<code>static int</code>	<code>INTEGER</code>	The constant in the Java programming language, sometimes referred to as a type code, that identifies the generic SQL type INTEGER.
<code>static int</code>	<code>NULL</code>	The constant in the Java programming language, sometimes referred to as a type code, that identifies the generic SQL type NULL.
<code>static int</code>	<code>REAL</code>	The constant in the Java programming language, sometimes referred to as a type code, that identifies the generic SQL type REAL.
<code>static int</code>	<code>TIME</code>	The constant in the Java programming language, sometimes referred to as a type code, that identifies the generic SQL type TIME.
<code>static int</code>	<code>VARCHAR</code>	The constant in the Java programming language, sometimes referred to as a type code, that identifies the generic SQL type VARCHAR.

##### Constructors

```
Types()
```

---

## Fields

### CHAR

```
public static final int CHAR
```

The constant in the Java programming language, sometimes referred to as a type code, that identifies the generic SQL type CHAR.

## **DATE**

```
public static final int DATE
```

The constant in the Java programming language, sometimes referred to as a type code, that identifies the generic SQL type DATE.

## **INTEGER**

```
public static final int INTEGER
```

The constant in the Java programming language, sometimes referred to as a type code, that identifies the generic SQL type INTEGER.

## **NULL**

```
public static final int NULL
```

The constant in the Java programming language, sometimes referred to as a type code, that identifies the generic SQL type NULL.

## **REAL**

```
public static final int REAL
```

The constant in the Java programming language, sometimes referred to as a type code, that identifies the generic SQL type REAL.

## **TIME**

```
public static final int TIME
```

The constant in the Java programming language, sometimes referred to as a type code, that identifies the generic SQL type TIME.

## **VARCHAR**

```
public static final int VARCHAR
```

The constant in the Java programming language, sometimes referred to as a type code, that identifies the generic SQL type VARCHAR.

---

## **Constructors**

### **Types()**

```
public Types()
```

rgma

## Units

### Declaration

```
public class Units
```

```
rgma.Units
```

### Description

Time units.

#### Member Summary

##### Fields

```
static Units DAYS
    Days.

static Units HOURS
    Hours.

static Units MILLIS
    Milliseconds (1/1000 S).

static Units MINUTES
    Minutes.

static Units SECONDS
    Seconds.
```

##### Methods

```
boolean equals(Units units)
    Compares this Units object with another.
```

---

### Fields

#### DAYS

```
public static final rgma.Units DAYS
    Days.
```

#### HOURS

```
public static final rgma.Units HOURS
    Hours.
```

#### MILLIS

```
public static final rgma.Units MILLIS
```

Milliseconds (1/1000 S).

## MINUTES

```
public static final rgma.Units MINUTES  
Minutes.
```

## SECONDS

```
public static final rgma.Units SECONDS  
Seconds.
```

---

## Methods

### **equals(Units)**

```
public boolean equals(rgma.Units units)  
Compares this Units object with another.
```

#### **Parameters:**

`units` - The other Units object.

#### **Returns:**

True if they represent the same units.

rgma

## UnknownResourceException

### Declaration

```
public class UnknownResourceException
```

```
rgma.UnknownResourceException
```

### Description

Exception thrown when a resource cannot be found in the resource framework.

#### Member Summary

##### Constructors

```
UnknownResourceException(String message)  
Constructs an UnknownResourceException object.
```

---

### Constructors

#### UnknownResourceException(String)

```
public UnknownResourceException(String message)
```

Constructs an UnknownResourceException object.

##### Parameters:

message - The error message.

rgma

## Predicate

### Declaration

```
public class Predicate extends SQLStatement
```

```
rgma.SQLStatement
|
+--rgma.Predicate
```

### Description

An SQL WHERE clause.

#### Member Summary

##### Constructors

```
Predicate()
    Creates a new empty Predicate.

Predicate(String predicate)
    Creates a new Predicate from a String of SQL.
```

#### Inherited Member Summary

##### Methods inherited from class SQLStatement

```
getStatement()
```

---

## Constructors

### Predicate()

```
public Predicate()
    Creates a new empty Predicate.
```

### Predicate(String)

```
public Predicate(String predicate)
    Creates a new Predicate from a String of SQL.
```

#### Parameters:

predicate - An SQL WHERE clause as a String.



## Computing Element

### Data Types (also used in WMS)

- GRID\_Job\_Description - JDL/ClassAd of the job, both at submission time (CondorG submit file) and after match/optimisation.
- GRID\_Job\_TQ\_ID - Unique ID obtained at submission time, and used to identify the job in the TQ. Would be equivalent to the AliEn TaskQueue Job Id and to the current EDG Job ID.
- GRID\_Job\_CE\_ID - Job ID used within the 'CE'. Would be equivalent to the current Alien QueueId, and to the CondorG job ID for EDG.
- GRID\_Oper\_Status - Generic error description.
- GRID\_Job\_Info - Set of 'useful' information about \*one\* job. Should contain be a combination of what AliEn 'top' and 'ps' return, of what is found in a line of the AliEn TaskQueue database, and iof what EDG get\_status and get\_logging\_info return.
- GRID\_Job\_Status - Some sort of enum (used in GRID\_Job\_Info) describing the possible states of a job. Purely as food for thought, here's the current possible states for AliEn, EDG, Condor:

AliEn:

```
WAITING, ASSIGNED, QUEUED, INSERTING, RUNNING,
EXPIRED, ERROR_S, ERROR_E, ERROR_A, ERROR_V, KILLED, DONE, IDLE,
INTERACTIV, FAULTY
```

EDG (from the JobStatus object):

```
enum Code {
  UNDEF = 0,          /**< indicates invalid, i.e. uninitialized instance
  */
  SUBMITTED,        /**< entered by the user to the User Interface or
  registered by Job Partitioner */
  WAITING,          /**< Accepted by WMS, waiting for resource allocation
  */
  READY,           /**< Matching resources found */
  SCHEDULED,       /**< Accepted by LRMS queue */
  RUNNING,         /**< Executable is running */
  DONE,           /**< Execution finished, output is available */
  CLEARED,         /**< Output transfered back to user and freed */
  ABORTED,         /**< Aborted by system (at any stage) */
  CANCELLED,       /**< Cancelled by user */
}
```

```
UNKNOWN,          /**< Status cannot be determined */
PURGED, /**< Job has been purged from bookkeeping server (for LB-
>RGMA interface) */
CODE_MAX
};
```

```
Condor:
char *JobStatusNames[] = {
    "UNEXPANDED",
    "IDLE",
    "RUNNING",
    "REMOVED",
    "COMPLETED",
    "HELD",
    "SUBMISSION_ERR",
};
```

#### **API:**

GRID\_Oper\_Status SubmitJob(GRID\_Job\_Description jdl, GRID\_Job\_CE\_ID \*id);  
Transfer job control to CE.

GRID\_Oper\_Status CancelJob(GRID\_Job\_CE\_ID id);  
Remove job from CE.

GRID\_Oper\_Status JobStatus(GRID\_Job\_CE\_ID id, GRID\_Job\_Info \*jobinfo);  
Get current status information.

## Workload Management

Note: job hold/resume and job signal functionality is at the moment left out.

GRID\_Oper\_Status InsertJob(GRID\_Job\_Description jdl, GRID\_Job\_TQ\_ID \*jobid);  
Insert job into TQ.

GRID\_Oper\_Status RemoveJob(GRID\_Job\_TQ\_ID jobid);  
Delete job from TQ (and CE if necessary).

GRID\_Oper\_Status JobStatus(GRID\_Job\_TQ\_ID jobid, GRID\_Job\_Info \*jobinfo);  
Get current status information.

GRID\_Oper\_Status GetJdl(GRID\_Job\_TQ\_ID jobid, GRID\_Job\_Description \*jdl);  
Obtain current job JDL.

GRID\_Oper\_Status SetJdl(GRID\_Job\_TQ\_ID jobid, GRID\_Job\_Description jdl);  
Replace job JDL in TQ.

## Storage Element

<b>Name</b>	<code>grid_filedesc_init</code>
<b>Synopsis</b>	Initialize a grid file descriptor
<b>Fields</b>	<code>grid_filedesc_t</code> File descriptor to initialize. <code>*fd</code>
<b>Errors</b>	<code>GRID_ENULL</code> The descriptor has not been allocated yet.
<b>Notes</b>	This has no correspondence in AliEn or EDG. The structure is used to store security information about the user, and session information for the Grid services.

<b>Type Name</b>	<code>grid_filedesc_t</code>
<b>Def</b>	<code>typedef struct</code> [to be detailed]
<b>Notes</b>	The file descriptor is a structure containing the necessary information to be able to carry out the file operations described below. Information like the user's credentials, the current position in the file and buffering information will be part of it.

<b>Name</b>	<code>grid_open</code>
<b>Synopsis</b>	Open a grid file
<b>Fields</b>	<code>char* path</code> Full logical file path of the file to open  <code>int flags</code> The open flags. Accepted are:  <code>GRID_O_RDONLY</code> Request file for read only. <code>GRID_O_WRONLY</code> Request file for write only. <code>GRID_O_APPEND</code> For write, don't overwrite but append to file  <code>GRID_O_CREAT</code> For write, if file does not exist, don't come back with an error but create the file.  <code>GRID_O_EXCL</code> If the file exists, when used with <code>GRID_O_CREAT</code> , an error will be returned.  <code>grid_filedesc_t</code> The file descriptor being filled in. <code>*fd</code>
<b>Errors</b>	<code>GRID_ENULL</code> The file handle structure being passed in is not initialized.  <code>GRID_ENEXIST</code> Returned on read: there is no such entry in the file catalog.

<b>Notes</b>	GRID_EUNKNOWNSURL	The SURL which is in the catalog is not recognized by the SE. This is an inconsistency between the catalog and the actual data on the SE.
	GRID_EACCES	The user is not allowed to do read/write to the path.
	GRID_ENAMETOOLONG	The pathname is too long.
	GRID_EISDIR	The pathname refers to a directory and not to a file.
	GRID_ENOTDIR	A pathname component is in fact not a directory.
	GRID_ENOENT	A pathname component (parent) does not exist.
	GRID_EEXIST	The pathname already exists and GRID_O_CREAT and GRID_O_EXCL were used.
	GRID_EDENIED	The user is not allowed to unregister.
<p>This corresponds to the open() call in AliEn and has no direct corresponding call in EDG, where this was done in a combination of replica manager/broker info (lookup of LFN) and system setup (NFS mount).</p> <p>Note also that for the time being there is no read-write mode. This is due to the distributed nature of the system and a simplification in semantics, taken from AliEn.</p>		

<b>Name</b>	<b>grid_close</b>	
<b>Synopsis</b>	Open a grid file	
<b>Fields</b>	grid_filedesc_t	File descriptor to close.
	*fd	
<b>Errors</b>	GRID_EBADF	Not a valid file descriptor.
<b>Notes</b>	This corresponds to the close() call in AliEn and has no direct corresponding call in EDG.	

<b>Name</b>	<b>grid_read</b>	
<b>Synopsis</b>	Read from a grid file. Reads are sequential.	
<b>Fields</b>	grid_filedesc_t	File descriptor. The file has to be opened for read.
	*fd	
	const void* buf	The buffer to be read to.
	size_t count	The amount of bytes to be read.
	size_t *count	The amount of bytes actually read.
<b>Errors</b>	GRID_EBADF	Not a valid file descriptor.

	GRID_EINVAL	The file descriptor given is attached to a file that cannot be written.
	GRID_EPIPE	While writing, the connection to the service hosting the file was lost.
<b>Notes</b>	This corresponds to the read() call in AliEn and has no direct corresponding call in EDG.	

<b>Name</b>	<b>grid_write</b>	
<b>Synopsis</b>	Write into a grid file. Writes are sequential.	
<b>Fields</b>	grid_filedesc_t *fd	File descriptor. The file has to be opened for write.
	const void* buf	The buffer to be written.
	size_t count	The amount of bytes to be written.
	size_t *count	The amount of bytes actually written.
<b>Errors</b>	GRID_EBADF	Not a valid file descriptor.
	GRID_EINVAL	The file descriptor given is attached to a file that cannot be written.
	GRID_EPIPE	While writing, the connection to the service hosting the file was lost.
<b>Notes</b>	This corresponds to the write() call in AliEn and has no direct corresponding call in EDG.	

<b>Name</b>	<b>grid_fseek</b>	
<b>Synopsis</b>	Position the read stream. Does not work for writes, works only forward for reads.	
<b>Fields</b>	grid_filedesc_t *fd	File descriptor. The file has to be opened for read.
	size_t count	The amount of bytes to skipped.
<b>Errors</b>	GRID_EBADF	Not a valid file descriptor.
	GRID_EINVAL	The file descriptor given is attached to a file that cannot be read (ie file has been opened for write).
	GRID_EPIPE	While seeking the connection to the service hosting the file was lost.
<b>Notes</b>	This has no corresponding call in AliEn and has no direct corresponding call in EDG.	

<b>Name</b>	<b>grid_stat</b>	
<b>Synopsis</b>	Return the status of a grid file based on its logical file name.	

<b>Fields</b>	<code>char *filename</code>	File descriptor. The file has to be opened for write.
	<code>struct grid_fstat_t* statbuf</code>	The filestat structure to be filled. It is described below.
	<b>Errors</b>	
	<code>GRID_ENEXIST</code>	There is no such entry in the file catalog.
	<code>GRID_EACCES</code>	The user is not allowed to do read this information.
	<code>GRID_ENAMETOOLONG</code>	The pathname is too long.
	<code>GRID_ENOTDIR</code>	A pathname component is in fact not a directory.
	<code>GRID_ENOENT</code>	A pathname component (parent) does not exist.
<b>Notes</b>	This corresponds to the <code>stat()</code> call in AliEn and has no direct corresponding call in EDG.	

<b>Type Name</b>	<code>grid_fstat_t</code>
<b>Def</b>	<code>typedef struct [to be detailed]</code>
<b>Notes</b>	The <code>fstat</code> structure will contain the detailed information about a grid file, like its GUID, its ACL, size and modification dates.

Note: All of the file system commands manipulating directories are described under the File Catalog component.

## File Transfer Service

<b>Name</b>	<b>grid_schedule_transfer</b>	
<b>Synopsis</b>	Put a new transfer request on the file transfer queue. This is a nonblocking call.	
<b>Fields</b>	<code>char * sourceURL</code>	The source file name. This needs to be a valid SURL recognized by an SE or a http, ftp or gsiftp URL.
	<code>char * destURL</code>	The destination file name. This also is a SURL recognized by an SE or a ftp or gsiftp URL.
	<code>grid_stid_t *id</code>	The scheduled transfer id returned.
<b>Errors</b>	<code>GRID_EBADSURL</code>	Not a valid SURL for either source or dest.
	<code>GRID_EACCESS</code>	The user is not allowed to write / read the file.
<b>Notes</b>		

<b>Name</b>	<b>grid_cancel_transfer</b>	
<b>Synopsis</b>	Cancel a transfer request. This will simply remove an entry from the queue. If the transfer is already in progress, this has no effect.	
<b>Fields</b>	<code>grid_stid_t id</code>	The transfer id to be cancelled.
<b>Errors</b>	<code>GRID_EBADTRANSID</code>	Not a valid transfer id (anymore).
	<code>GRID_EACCESS</code>	The user is not allowed to cancel the transfer.
<b>Notes</b>		

<b>Name</b>	<b>grid_transfer_status</b>	
<b>Synopsis</b>	Show the status of the transfer.	
<b>Fields</b>	<code>grid_stid_t id</code>	The transfer id to listed.
	<code>int *stat</code>	The status. It can be <code>GRID_ST_PENDING</code> , <code>GRID_ST_TRANSFERRING</code> , <code>GRID_ST_CANCELLED</code> , <code>GRID_ST_DONE</code> .
<b>Errors</b>	<code>GRID_EBADTRANSID</code>	Not a valid transfer id.
	<code>GRID_EACCESS</code>	The user is not allowed to read the status.
<b>Notes</b>		

<b>Name</b>	<b>grid_transfer_list</b>	
<b>Synopsis</b>	Show the list of scheduled transfers. Only those transfers are listed that the user is allowed to see. This call can be called repeatedly to get the next entry from the stream. It will return GRID_EOF if there are no more entries left to list.	
<b>Fields</b>	grid_stid_t *id	The transfer id of the entry.
	char *source	The source to be transferred from. This is a buffer which should be allocated by the user before calling the method.
	char *dest	The destination to transfer to. This is also a buffer, just like source.
	Int *stat	The status. It can be GRID_ST_PENDING, GRID_ST_TRANSFERRING, GRID_ST_CANCELLED, GRID_ST_DONE.
<b>Errors</b>	GRID_EOF	This is not really an error, it indicates that there are no more entries to be retrieved. The source and dest buffers and id, stat fields are unchanged if EOF is reached.
	GRID_ERANGE	The source or dest buffer is not big enough to store the entry. There is a field GRID_NAME_MAX that defines the maximal name length.
	GRID_EACCESS	The user is not allowed to list the status.
<b>Notes</b>		

## File Placement Service

<b>Name</b>	<b>grid_create_replica</b>	
<b>Synopsis</b>	Create a replica at a given location. This will submit and monitor a file transfer to the file transfer service and update the file catalog accordingly upon successful replication. This is a blocking call until a replication and registration is complete.	
<b>Fields</b>	<code>char *path</code>	The full logical file name path of the file of which a new replica is to be created.
	<code>char* destination</code>	The destination to replicate to. This can be a fully qualified SURL or just the SE to replicate to.
<b>Errors</b>	<code>GRID_EEXIST</code>	There is already a replica at the given destination.
	<code>GRID_ENEXIST</code>	The path does not exist in the catalog.
	<code>GRID_EACCESS</code>	The user is not allowed to create a new replica at the given destination.
	<code>GRID_ENAMETOOLONG</code>	The pathname is too long.
	<code>GRID_ENOENT</code>	A pathname component (parent) does not exist.
<b>Notes</b>	The source to replicate from will be chosen by the service. Trivially it will choose a file from the local store if one is available, otherwise it chooses one at random. This corresponds to the createMirror call in AliEn and the replica manager replicateFile command in EDG. In EDG the source was chosen based on some networking monitoring metrics.	

<b>Name</b>	<b>grid_put_file</b>	
<b>Synopsis</b>	Put a new file into the grid explicitly. The source file can exist either on the local file system or is accessible through one of the http, ftp, gsiftp protocols. This is a blocking call for local files and nonblocking for other files where the transfer service will be used to perform the put. This call also has a bulk operation corresponding call, <code>grid_bulk_put_file</code> .	
<b>Fields</b>	<code>char *path</code>	The full logical file name path of the file that should exist from now on in the grid.
	<code>char *sourceURL</code>	The source URL. This has to be a valid url with schema file, http, ftp, gsiftp.
	<code>char *destSURL</code>	This is optional and can be left NULL, in which case the service will choose a name to put the file to on the local SE. Otherwise it tries to create the file using the given SURL.

<b>Errors</b>	GRID_EEXIST	There is already a file in the catalog with the given path.
	GRID_ENEXIST	The source does not exist or is not accessible.
	GRID_EACCESS	The user is not allowed to create a new file in the given path or the given destSURL.
	GRID_ENAMETOOLONG	The pathname is too long.
	GRID_ENOENT	A pathname component (parent) does not exist.
	GRID_EINVSURL	The SURL for destination given is invalid
	GRID_ESURLEXIST	The dest SURL already exists on the SE
<b>Notes</b>	This corresponds to the copyAndRegister call in the EDG replica catalog.	

<b>Name</b>	<b>grid_bulk_put_file</b>	
<b>Synopsis</b>	Put a set of new files into the grid. The set of files is defined in the XML string. It may contain also metadata that is to be added to each file as well as ACLs.	
<b>Fields</b>	char *xml	The XML string specifying the files and all attributes thereof to be put. [to be detailed]
	int policy	Flags to specify the failure policy. The options are: GRID_FAIL_ANY Fail for all if any of the files cannot be created. GRID_FAIL_ALL Fail only if none of the files can be put. The result string will contain eventual individual failures.
	char *resultXML	Report on the bulk operation. This is also an XML string, containing the pathnames and individual return codes. The return codes are listed below.
<b>Errors</b>	GRID_EEXIST	There is already a file in the catalog with the given path.
	GRID_ENEXIST	The source does not exist or is not accessible.
	GRID_EACCESS	The user is not allowed to create a new file in the given path or the given destSURL.
	GRID_ENAMETOOLONG	The pathname is too long.
	GRID_ENOENT	A pathname component (parent) does not exist.
	GRID_EINVSURL	The SURL for destination given is invalid
	GRID_ESURLEXIST	The dest SURL already exists on the SE
<b>Notes</b>	This corresponds to the bulkCopyAndRegister call in the EDG replica catalog. It is first checked whether the user has the right to create all logical names in the catalog before starting the copy operations through the file transfer service.	

<b>Name</b>	<b>grid_delete_replica</b>	
<b>Synopsis</b>	Remove a replica from a given location. This will submit a removal request to the SE and remove the entry from the catalog.	
<b>Fields</b>	char *path	The full logical file name path of the file of which a replica is to be removed.
	char* replica	The replica to remove. This can be a fully qualified SURL or just the SE to remove from.
<b>Errors</b>	GRID_ENEXIST	There is no such path in the catalog.
	GRID_EINVAL	There is no such replica to delete.
	GRID_EACCESS	The user is not allowed to create a new replica at the given destination.
	GRID_ENAMETOOLONG	The pathname is too long.
	GRID_ENOENT	A pathname component (parent) does not exist.
	GRID_EINVSURL	The SURL is invalid.
	GRID_EFAIL	The SURL could not be removed from the SE. This occurs if there is an inconsistency between the catalog and the SE. The catalog entry is removed!
<b>Notes</b>	It is first checked whether the user is allowed to perform the operation on the given SURL. The entry is removed from the catalog first, and only if that operation is successful will a removal request be made at the SE (which should not but can fail with GRID_EFAIL). If this was the last replica in the catalog, the logical entry is being kept, but it will have no actual replicas associated with it and the file size is reset to 0.	

## File and Replica Catalog

### Basic types

C data types were chosen for interface definitions but with the underlying assumption that all of them could be mapped to XMLSchema 1.0 types as defined in <http://www.w3.org/TR/xmlschema-2/#built-in-datatypes>, including complex structures obeying this standard.

### Logical directory methods

These methods enable interaction with the logical namespace of the catalog.

<b>Name</b>	<b>grid_readdir</b>
<b>Synopsis</b>	Return the next entry in the directory stream.
<b>Fields</b>	<code>char*</code> directory Directory to list <code>char*</code> buf Buffer to place next entry in.
<b>Errors</b>	GRID_EBADF Not a valid directory GRID_EOF This is not really an error, it indicates that there are no more entries to be retrieved or that the directory is empty. The buffer is unchanged if EOF is reached. GRID_ERANGE The buffer is not big enough to store the entry. There is a field <code>GRID_NAME_MAX</code> that defines the maximal name length.
<b>Notes</b>	The behavior is similar but not identical to the posix <code>readdir</code> system call. This corresponds to the <code>readdir()</code> call in AliEn and has no correspondent in EDG.

<b>Name</b>	<b>grid_mkdir</b>
<b>Synopsis</b>	Create a new logical directory.
<b>Fields</b>	<code>char*</code> directory Full path of the directory to create
<b>Errors</b>	GRID_EEXIST There is already an entry in the catalog with this name (either file or directory) GRID_EACCES The parent directory does not allow the user to create a directory (see posix acl semantics below) GRID_ENAMETOOLONG The pathname is too long. GRID_ENOENT A pathname component (parent) does not exist.
<b>Notes</b>	This corresponds to the <code>MkDir()</code> call in AliEn and has no correspondent in EDG.

<b>Name</b>	<b>grid_rmdir</b>	
<b>Synopsis</b>	Delete a logical directory.	
<b>Fields</b>	char* directory	Full path of the directory to delete
<b>Errors</b>	GRID_EEXIST	There is already an entry in the catalog with this name (either file or directory)
	GRID_EACCES	The parent directory does not allow the user to remove a directory or directory not readable (see posix acl semantics below)
	GRID_ENAMETOOLONG	The pathname is too long.
	GRID_ENOENT	A pathname component (parent) does not exist.
	GRID_ENOTEMPTY	The directory is not empty
	GRID_ENOTDIR	The path is actually not a directory
<b>Notes</b>	This corresponds to the Rmdir() call in AliEn and has no correspondent in EDG.	

#### Virtual directory methods

<b>Name</b>	<b>grid_mkvdire</b>	
<b>Synopsis</b>	Create a new virtual directory.	
<b>Fields</b>	char* directory	Full path of the directory to create
	char* queryXML	The XML which defines the query and the query and directory metadata. It contains the directives for example about the maximal cardinality of the result, eventual offset, etc.
<b>Errors</b>	GRID_EEXIST	There is already an entry in the catalog with this name (either file or directory)
	GRID_EACCES	The parent directory does not allow the user to create a directory (see posix acl semantics below)
	GRID_ENAMETOOLONG	The pathname is too long.
	GRID_ENOENT	A pathname component (parent) does not exist.
	GRID_EBADQUERY	The query has failed to create the virtual directory.
<b>Notes</b>	This has no corresponding call in AliEn or EDG.	

<b>Name</b>	<b>grid_vdir_refresh</b>
<b>Synopsis</b>	Refresh a virtual directory.

<b>Fields</b>	char* directory	Full path of the directory to refresh.
<b>Errors</b>	GRID_ENEXIST	There is no such entry in the catalog.
	GRID_EACCES	The user is not allowed to refresh the virtual directory.
	GRID_ENAMETOOLONG	The pathname is too long.
	GRID_ENOENT	A pathname component (parent) does not exist.
	GRID_EBADQUERY	The query has failed to update the virtual directory.
<b>Notes</b>	There is no corresponding call in AliEn or EDG.	

<b>Name</b>	<b>grid_rmmdir</b>	
<b>Synopsis</b>	Remove a virtual directory.	
<b>Fields</b>	char* directory	Full path of the directory to remove.
<b>Errors</b>	GRID_ENEXIST	There is no such entry in the catalog.
	GRID_EACCES	The user is not allowed to remove the virtual directory.
	GRID_ENAMETOOLONG	The pathname is too long.
	GRID_ENOENT	A pathname component (parent) does not exist.
	<b>Notes</b>	There is no corresponding call in AliEn or EDG.

### Security methods

[to be completed in detail, including structures]

grid\_acl\_set\_file

grid\_acl\_get\_file

grid\_acl\_copy\_entry, grid\_acl\_create\_entry, grid\_acl\_delete\_entry,

grid\_acl\_get\_entry, grid\_acl\_valid

grid\_acl\_add\_perm, grid\_acl\_calc\_mask, grid\_acl\_clear\_perms,

grid\_acl\_delete\_perm, grid\_acl\_get\_permset, grid\_acl\_set\_permset

grid\_acl\_get\_qualifier, grid\_acl\_get\_tag\_type, grid\_acl\_set\_qualifier,

grid\_acl\_set\_tag\_type

grid\_acl\_copy\_entry, grid\_acl\_copy\_ext, grid\_acl\_from\_text,

grid\_acl\_to\_text, grid\_acl\_size

## Logical files

<b>Name</b>	<b>grid_create</b>	
<b>Synopsis</b>	Create a new logical file name entry without actual associated data.	
<b>Fields</b>	char* path	Full logical file path of the file to register
<b>Errors</b>	GRID_EEXIST	There is already an entry in the file catalog with this name
	GRID_EACCES	The parent directory does not allow the user to write a new entry here.
	GRID_ENAMETOOLONG	The pathname is too long.
	GRID_ENOENT	A pathname component (parent) does not exist.
<b>Notes</b>	This does not correspond to anything in AliEn or EDG.	

<b>Name</b>	<b>grid_delete</b>	
<b>Synopsis</b>	Delete logical file name entry. This will not remove the files from storage, use <code>grid_delete_replica</code> or <code>grid_delete_all</code> instead. This is purely a catalog operation.	
<b>Fields</b>	char* path	Full logical file path of the file to delete.
<b>Errors</b>	GRID_ENEXIST	There no such entry in the file catalog.
	GRID_EACCES	The parent directory does not allow the user to delete.
	GRID_ENAMETOOLONG	The pathname is too long.
	GRID_ENOENT	A pathname component (parent) does not exist.
<b>Notes</b>		

<b>Name</b>	<b>grid_rename</b>	
<b>Synopsis</b>	Rename the logical file name.	
<b>Fields</b>	char* path	Full logical file path of the file to rename
	char* newpath	Full logical file path of the new file name
<b>Errors</b>	GRID_EEXIST	There is already an entry in the file catalog with this name (new name)
	GRID_ENEXIST	There is no such path in the catalog (existing name)
	GRID_EACCES	The parent directory does not allow the user to remove the old entry or to write the new entry.
	GRID_ENAMETOOLONG	The pathname is too long.
	GRID_ENOENT	A pathname component (parent) does not exist.

## Notes

### Symbolic links

<b>Name</b>	<b>grid_symlink</b>	
<b>Synopsis</b>	Create a symbolic link.	
<b>Fields</b>	char* path	Full logical file path of the new file name (i.e. the symlink)
	char* linkpath	Full logical file path of existing file to link to
<b>Errors</b>	GRID_EEXIST	There is already an entry in the file catalog with this name (symlink path, first arg)
	GRID_ENEXIST	There is no such path in the catalog (existing name to link to, second arg)
	GRID_EACCES	The parent directory does not allow the user to write the entry.
	GRID_ENAMETOOLONG	The pathname is too long.
	GRID_ENOENT	A pathname component (parent) does not exist.
<b>Notes</b>		

### Replica manipulation and entry creation

These methods are all catalog-only methods and do not involve data movement. See above under 'data management' for the methods involving both registration and data movement.

<b>Name</b>	<b>grid_register</b>	
<b>Synopsis</b>	Register a file which exists on a SE.	
<b>Fields</b>	char* path	Full logical file path of the file to register
	char* SURL	The SURL of the
<b>Errors</b>	GRID_EEXIST	There is already an entry in the file catalog with this name
	GRID_EACCES	The parent directory does not allow the user to write a new entry here.
	GRID_ENAMETOOLONG	The pathname is too long.
	GRID_ENOENT	A pathname component (parent) does not exist.
	GRID_EINVSURL	The SURL given is invalid
	GRID_ESURLNEXIST	The SURL does not exist on the SE
	GRID_EDENIED	The user is not allowed to register a new replica.

<b>Notes</b>	This corresponds to the RegisterFile() call in AliEn and the replica manager registerFile method in EDG.
--------------	--

<b>Name</b>	<b>grid_list_replicas</b>
<b>Synopsis</b>	List all known replicas of a file.
<b>Fields</b>	char* path Full logical file path of the file char* buf Buffer to place next SURL in.
<b>Errors</b>	GRID_ENEXIST No such file. GRID_EOF This is not really an error, it indicates that there are no more entries to be retrieved. The buffer is unchanged if EOF is reached. GRID_ERANGE The buffer is not big enough to store the entry. There is a field GRID_NAME_MAX that defines the maximal name length. GRID_EACCES The user is not allowed to read the path. GRID_ENAMETOOLONG The pathname is too long. GRID_ENOENT A pathname component (parent) does not exist. GRID_EDENIED The user is not allowed to list replicas.
<b>Notes</b>	This corresponds to the replica manager listReplicas method in EDG. It has similar semantic behavior as the grid_readdir command.

<b>Name</b>	<b>grid_unregister</b>
<b>Synopsis</b>	Un-register a file replica
<b>Fields</b>	char* path Full logical file path of the file modify the registrations char* SURL The SURL of the PFN to be unregistered.
<b>Errors</b>	GRID_ENEXIST There is no such entry in the file catalog. GRID_EACCES The user is not allowed to do read the path. GRID_ENAMETOOLONG The pathname is too long. GRID_ENOENT A pathname component (parent) does not exist. GRID_EINVSURL The SURL given is invalid GRID_EDENIED The user is not allowed to unregister.
<b>Notes</b>	This corresponds to no call in AliEn and the replica manager unregisterFile method in EDG. It does not remove the LFN. If the last entry has been removed, the LFN is still not removed, it simply has no replica.

Sessions

The methods presented here run in a context of a session. This implies that for every call *current directory* is defined, as well as *root directory*.

Many methods derive their options sets from Posix originals, many may support omnipresent grid options like asynchronous operation, publishing or others. [The details are up for discussion.]

Definitions of some of these grid options follow pseudo EBNF notation:

grid\_option:

```
    grid_async_option  |
    publish_option     |
    subscribe_option
    ;
```

async\_option :

```
    |
    ASYNC_FLAG timeout_option return_method action
    ;
```

publish\_option:

```
    |
    PUBLISH_FLAG metadata_tags
    ;
```

subscribe\_option:

```
    |
    SUBSCRIBE_FLAG metadata_tags
    ;
```

Action:

```
    |
    ACTION_FLAGS Method {call(Method(...)};
```

timeout\_option:

```
{default}
    NUMBER SCALE //when timeout occurs in SCALE units
    ;
```

return\_method:

```
{default acknowledgment} |
```

```

EMAIL email_ops    |
QUEUE queueops     |
FILECREATION filecreatops
; //this also could be a list (mixture any of these)

```

Most methods in the complex and basic category can be extended to include sessions.

### Complex methods

The complex operations involve the operations that do more than just atomic calls to the catalog. They operate also not just on simple types but on the structures that we define below. The methods involving session handling also are in the category of complex operations.

### Methods involving metadata

The metadata calls are routed through to the metadata API described below.

API call	Description
grid_register_with_md	Register a file with a set of metadata in one go
grid_fc_add_tag	Add a tag of a certain type to a given file
grid_fc_remove_tag	Remove the tag from the file
grid_fc_show_tags	Show tags and their type associated with a file.
grid_fc_tag_exists	Ask whether a tag exists for a given file
grid_fc_set_tag_value	Set the value of a given tag of a file
grid_fc_get_tag_value	Retrieve the value of a tag of a file
grid_fc_list_files_by_tag	List all files having a given value for a tag in a directory, with offset and number of results to return
grid_fc_get_files_by_tag_search	Find all files matching a tag query, starting from a given directory, with offset and number of results to return.

### Session-specific calls

```

grid_cwdir
grid_chdir
grid_pwd

```

### Middleware-Domain API

The job scheduler will need to know all available replicas (SURLs) based on a (set of) LFNs. This operation has to be very efficient and has to scale very well.

[GUID-related calls – to be included]

### Structures

[These need to be detailed]

<b>Name</b>	grid_session_t
<b>Synopsis:</b>	The main predefined user structure is used to store context of the

	user session operations, like the current directory. By definition, this is a singleton object per user session. Used by external programs and users to store context of a user session.	
<b>Fields</b>	long sessionId	Unique session ID
	string currentDirectory	Context of the session.
	string homeDirectory	Same as in u-space in unix
	long sessionLifetime	End-of-session lifetime, sessions need to be able to time out upon client failures or idleness
<b>Notes</b>	Sessions are always tightly coupled with the user's security object.	

<b>Name</b>	grid_user_t	
<b>Synopsis:</b>	The user's identity object based on the credentials.	
<b>optional</b>	string userDN	The user's distinguished name
	string[] userRoles	Roles the user has been associated with
	long validityTime	
	long creationTime	
<b>Notes:</b>		

<b>Name</b>	grid_file_t	
<b>Synopsis:</b>	The main predefined user structure is used to store information on a file.	
<b>Fields</b>	String LFN	Full path of the file
	ACL accessControl	ACL of the file
	long size	File size
	long creationTime	Creation time of file
	URI[] SURList	List of replicas
	URI GUID	GUID of file
	URI parent	GUID of parent directory
	FCMetadata md	Metadata on file
<b>Notes:</b>	The fields do not need to be present or filled for some methods that only deal with simple lookups. There are specific methods that will fill in the fields of existing FCFile objects.	

<b>Name</b>	grid_directory_t	
<b>Synopsis:</b>	Specifies the nature of a directory. It can be a virtual directory or a 'real' logical directory.	
<b>Fields</b>	string path	Full path of directory
	URI GUID	GUID of directory
	URI parent	GUID of parent directory

	ACL accessControl	The access control list of the directory
	boolean virtual	The virtual directory flag
	string query	Query string for virtual directories
	long maxsize	The maximal number of files in this directory
	long creationTime	Time when directory was created
	long refreshTime	Time when directory was last refreshed (query was rerun for virtual directories or last written to for logical directories)
<b>Notes:</b>		

<b>Name</b>	grid_metadata_t	
<b>Synopsis</b>	All metadata associated with a catalogue entry as returned through the metadata interface.	
<b>Fields</b>	long size	Number of key-value pairs for the entry
	URI GUID	The GUID of the LFN this metadata belongs to
	string metadata	An XML string with all the metadata filled.

## Posix ACLs

The following is taken from the Posix ACL man pages.

### ACL Types

Every object can be thought of as having associated with it an ACL that governs the discretionary access to that object; this ACL is referred to as an access ACL. In addition, a directory may have an associated ACL that governs the initial access ACL for objects created within that directory; this ACL is referred to as a default ACL.

### ACL Entries

An ACL consists of a set of ACL entries. An ACL entry specifies the access permissions on the associated object for an individual user or a group of users as a combination of read, write and search/execute permissions.

An ACL entry contains an entry tag type, an optional entry tag qualifier, and a set of permissions. We use the term qualifier to denote the entry tag qualifier of an ACL entry.

The qualifier denotes the identifier of a user or a group, for entries with tag types of GRID\_ACL\_USER or GRID\_ACL\_GROUP, respectively. Entries with tag types other than GRID\_ACL\_USER or GRID\_ACL\_GROUP have no defined qualifiers. The following entry tag types are defined:

GRID_ACL_USER_OBJ	The GRID_ACL_USER_OBJ entry denotes access rights for the file owner.
GRID_ACL_USER	GRID_ACL_USER entries denote access rights for users identified by the entry's qualifier.
GRID_ACL_GROUP_OBJ	The GRID_ACL_GROUP_OBJ entry denotes access rights for the file group.
GRID_ACL_GROUP	GRID_ACL_USER entries denote access rights for groups identified by the entry's qualifier.
GRID_ACL_MASK	The GRID_ACL_MASK entry denotes the maximum access rights that can be granted by entries of type GRID_ACL_USER, GRID_ACL_GROUP_OBJ, or GRID_ACL_GROUP.
GRID_ACL_OTHER	The GRID_ACL_OTHER entry denotes access rights for processes that do not match any other entry in the ACL.

When an access check is performed, the GRID\_ACL\_USER\_OBJ and GRID\_ACL\_USER entries are tested against the effective user ID. The effective group ID, as well as all supplementary group IDs are tested against the GRID\_ACL\_GROUP\_OBJ and GRID\_ACL\_GROUP entries.

### Valid ACLs

A valid ACL contains exactly one entry with each of the GRID\_ACL\_USER\_OBJ, GRID\_ACL\_GROUP\_OBJ, and GRID\_ACL\_OTHER tag types. Entries with

GRID\_ACL\_USER and GRID\_ACL\_GROUP tag types may appear zero or more times in an ACL. An ACL that contains entries of GRID\_ACL\_USER or GRID\_ACL\_GROUP tag types must contain exactly one entry of the GRID\_ACL\_MASK tag type. If an ACL contains no entries of GRID\_ACL\_USER or GRID\_ACL\_GROUP tag types, the GRID\_ACL\_MASK entry is optional.

All user ID qualifiers must be unique among all entries of GRID\_ACL\_USER tag type, and all group IDs must be unique among all entries of GRID\_ACL\_GROUP tag type.

The `grid_acl_get_file()` function returns an ACL with zero ACL entries as the default ACL of a directory, if the directory is not associated with a default ACL. The `grid_acl_set_file()` function also accepts an ACL with zero ACL entries as a valid default ACL for directories, denoting that the directory shall not be associated with a default ACL. This is equivalent to using the `grid_acl_delete_def_file()` function.

## Correspondence Between ACL Entries and File Permissions

The permissions defined by ACLs are a superset of the permissions specified by the file permission bits. The permissions defined for the file owner correspond to the permissions of the GRID\_ACL\_USER\_OBJ entry. The permissions defined for the file group correspond to the permissions of the GRID\_ACL\_GROUP\_OBJ entry, if the ACL has no GRID\_ACL\_MASK entry. If the ACL has an GRID\_ACL\_MASK entry, then the permissions defined for the file group correspond to the permissions of the GRID\_ACL\_MASK entry. The permissions defined for the other class correspond to the permissions of the GRID\_ACL\_OTHER\_OBJ entry.

Modification of the file permission bits results in the modification of the permissions in the associated ACL entries. Modification of the permissions in the ACL entries results in the modification of the file permission bits.

## Object Creation and Default ACLs

The access ACL of a file object is initialized when the object is created with `grid_mkdir()`, `grid_register()`, `grid_symlink()` functions. If a default ACL is associated with a directory, the default ACL of the directory is used to determine the ACL of the new object, i.e. the new object inherits the default ACL of the containing directory as its access ACL.

The new object is assigned an access ACL containing entries of tag types GRID\_ACL\_USER\_OBJ, GRID\_ACL\_GROUP\_OBJ, and GRID\_ACL\_MASK. The permissions of these entries are set to the permissions specified by the file creation mask.

## Access Check Algorithm

A process may request read, write, or execute/search access to a file object protected by an ACL. The access check algorithm determines whether access to the object will be granted.

1. **If** the effective user ID of the process matches the user ID of the file object owner, **then if** the GRID\_ACL\_USER\_OBJ entry contains the requested permissions, access is granted, **else** access is denied.
2. **Else if** the effective user ID of the process matches the qualifier of any entry of type GRID\_ACL\_USER, **then if** the matching GRID\_ACL\_USER entry and the GRID\_ACL\_MASK entry contain the requested permissions, access is granted, **else** access is denied.

3. **Else if** the effective group ID or any of the supplementary group IDs of the process match the qualifier of any entry of type `GRID_ACL_GROUP`, **then if** the `GRID_ACL_MASK` entry and any of the matching `GRID_ACL_GROUP` group entries contain the requested permissions, access is granted, **else** access is denied.
4. **Else if** the `GRID_ACL_OTHER` entry contains the requested permissions, access is granted.
5. **Else** access is denied.

## Acl Text Forms

A long and a short text form for representing ACLs is defined. In both forms, ACL entries are represented as three colon separated fields: an ACL entry tag type, an ACL entry qualifier, and the discretionary access permissions. The first field contains one of the following entry tag type keywords:

- |       |  |
|-------|--|
| user  | A user ACL entry specifies the access granted to either the file owner (entry tag type <code>GRID_ACL_USER_OBJ</code> ) or a specified user (entry tag type <code>GRID_ACL_USER</code> ).      |
| group | A group ACL entry specifies the access granted to either the file group (entry tag type <code>GRID_ACL_GROUP_OBJ</code> ) or a specified group (entry tag type <code>GRID_ACL_GROUP</code> ).  |
| mask  | A mask ACL entry specifies the maximum access which can be granted by any ACL entry except the user entry for the file owner and the other entry (entry tag type <code>GRID_ACL_MASK</code> ). |
| other | An other ACL entry specifies the access granted to any process that does not match any user or group ACL entries (entry tag type <code>GRID_ACL_OTHER</code> ).                                |

The second field contains the user or group identifier of the user or group associated with the ACL entry for entries of entry tag type `GRID_ACL_USER` or `GRID_ACL_GROUP`, and is empty for all other entries. A user identifier can be a user name or a user ID number in decimal form. A group identifier can be a group name or a group ID number in decimal form. The third field contains the discretionary access permissions. The read, write and search/execute permissions are represented by the `r`, `w`, and `x` characters, in this order. Each of these characters is replaced by the `-` character to denote that a permission is absent in the ACL entry. When converting from the text form to the internal representation, permissions that are absent need not be specified.

White space is permitted at the beginning and end of each ACL entry, and immediately before and after a field separator (the colon character).

### *Long Text Form*

The long text form contains one ACL entry per line. In addition, a number sign (`#`) may start a comment that extends until the end of the line. If an `GRID_ACL_USER`, `GRID_ACL_GROUP_OBJ` or `GRID_ACL_GROUP` ACL entry contains permissions that are not also contained in the `GRID_ACL_MASK` entry, the entry is followed by a number sign, the string "effective:", and the effective access permissions defined by that entry. This is an example of the long text form:

```

user::rw-
user:lisa:rw-      #effective:r-
group::r-
group:toolies:rw-  #effective:r-
mask::r-
```

```
other::r-
```

### *Short Text Form*

The short text form is a sequence of ACL entries separated by commas, and used for input. Comments are not supported. Entry tag type keywords may either appear in their full unabbreviated form, or in their single letter abbreviated form. The abbreviation for user is `u`, the abbreviation for group is `g`, the abbreviation for mask is `m`, and the abbreviation for other is `o`. The permissions may contain at most one each of the following characters in any order: `r`, `w`, `x` `-`. These are examples of the short text form:

```
u::rw-,u:lisa:rw-,g::r--,g:toolies:rw-,m::r--,o::r--  
g:toolies:rw,u:lisa:rw,u::wr,g::r,o::r,m::r
```

## Rationale

IEEE 1003.1e draft 17 defines Access Control Lists that include entries tag type `ACL_MASK`, and defines a mapping between file permission bits that is not constant. The standard working group defined this relatively complex interface in order to ensure that applications that are compliant with IEEE 1003.1 ("POSIX.1") will still function as expected on systems with ACLs. The IEEE 1003.1e draft 17 contains the rationale for choosing this interface in section B.23.

In addition to these, the mode parameter is dropped for the grid adaptation since we will enforce a default ACL at all times and in the client-service model the process ACL concept is dropped.

## Changes to the File Utilities

On a system that supports ACLs, the file utilities `ls`, `cp` and `mv` change their behavior in the following way:

- For files that have a default ACL or an access ACL that contains more than the three required ACL entries, the `ls` utility in the long form produced by `ls -l` displays a plus sign (+) after the permission string.
- If the `-p` flag is specified, the `cp` utility also preserves ACLs. If this is not possible, a warning is produced.
- The `mv` utility always preserves ACLs. If this is not possible, a warning is produced.

## Standards

The IEEE 1003.1e draft 17 ("POSIX.1e") document describes several security extensions to the IEEE 1003.1 standard. While the work on 1003.1e has been abandoned, many UNIX style systems implement parts of POSIX.1e draft 17, or of earlier drafts.

Linux Access Control Lists implement the full set of functions and utilities defined for Access Control Lists in POSIX.1e, and several extensions. The implementation is fully compliant with POSIX.1e draft 17; extensions are marked as such.

See also <http://www.guug.de/~winni/posix.1e/download.html>