



IST-2001-32133

GridLab - A Grid Application Toolkit and Testbed

Deliverable D8.9 - First release of DM services

Author(s):	Felix Hupfeld, Andrei Hutanu, Andre Merzky, Thorsten Schütt, Brygg Ullmer
Document Filename:	
Work package:	WP 8 - Data Handling and Visualization
Partner(s):	
Lead Partner:	ZIB
Config ID:	GridLab-08-DATA-0006-M24
Document classification:	Internal

Abstract: This document describes the APIs for WP8's high level data access services and libraries. These prototypes have been deployed on the GridLab testbed.



Contents

1	Introduction	2
2	Metadata Catalog	2
2.1	Architecture	2
2.2	Interface	2
2.3	Relations to other workpackages	5
2.3.1	WP9 - Portal	5
2.3.2	WP12 - Messaging	5
2.4	Planned Improvements	5
2.4.1	Query language	6
2.4.2	Schema Management	6
3	Remote Partial File Access	6
4	Replica Catalog	6
4.1	Namespace for logical file names	6
4.2	Database for replica locations	6
4.3	Interface to File Movement	7
4.4	Integration with other Services	7
5	File movement	7
5.1	ns__FileTask Struct Reference	7
5.2	Function Documentation	8
6	File browsing	12
6.1	Directory_entry Struct Reference	13
6.2	Directory_entries Struct Reference	13
6.3	List_response Struct Reference	13
6.4	List_structured_response Struct Reference	13
6.5	Size_response Struct Reference	14
6.6	Time_response Struct Reference	14
6.7	Function Documentation	14

1 Introduction

This deliverable produces a report on the first release of the data handling tools, and their integration and usability in testbed and application environments.

2 Metadata Catalog

2.1 Architecture

The services and infrastructure of a Grid host a number of persistent and semi-persistent *objects* like files, jobs and messages. These objects are scattered all over the Grid; their unifying property is that they can be named, ie. they have a unique identifier, preferably in URI form.

In order to be able to access these objects, the Grid user needs an interface to search this object space for objects of interest. Having found the wanted objects, the user can locate and access them.

This functionality is provided by the *Metadata Catalog* which acts as the Grid's interface for searching for Grid objects. These objects are registered in the Metadata Catalog along with their metadata automatically by other services or manually by a user. The registered *metadata* is an extract of the object's properties and content and acts as a descriptive representation of the object. This representation is formatted as attribute-value pairs and can be searched by clients of the service.

The Metadata Catalog is organized as a set of *namespaces*, each of which is a closed compartment that contains the metadata for a set of objects. Users that authenticate with the Metadata Catalog have certain access rights to a namespace and its objects. These access rights are specified by the *access policy* for the respective namespace.

Each user owns an associated *private namespace* whose access policy grants access only to the owner of these resources. This namespace can be used to register private objects of the user or to add private metadata to visible objects of other namespaces. This namespace can be referred to by the namespace name alias "Private."

The Metadata Catalog usually keeps one *public namespace* whose access policy grants read access to all authenticated users, but limits write access to owners of the respective objects. This namespace is called "Public."

Further namespaces – for example, supporting *collaboration in closed user groups* – can be created and given an access policy.

2.2 Interface

The Metadata Catalog is accessible as a web service. Authentication is done with TLS/SSL or GSI. This web service interface implements the following functions.

void createObject (in string *namespace*, in string *object*, out string *retcode*)

Create a new object and establish the necessary access control information so that the user owns the object.

Parameters:

namespace The namespace in which the object should be created.

object The object's identifier

Returns:

200 Object created
201 Made public
403 User not authorized
405 Object already exists
503 Service Unavailable

void deleteObject (in string *namespace*, in string *object*, out string *retcode*)

Delete the object in the given namespace.

Parameters:

namespace The namespace in which the object should be created.
object The object's identifier

Returns:

200 Object deleted
403 User not authorized
404 Object not found
503 Service Unavailable

void addAttribute (in string *namespace*, in string *object*, in string *attribute*, in string *value*, out string *retcode*)

Add an attribute to the given object.

Parameters:

namespace The namespace to which the value should be added.
object The object's identifier
attribute The name of the attribute
value The value of the attribute

Returns:

200 Value added
403 User not authorized
404 Object not found (create it first)
406 Attribute name not found (Attribute does not fit to scheme)
409 Value wrong type (like adding a string to a number attribute)
412 Only set operation is allowed
503 Service Unavailable

void setAttribute (in string *namespace*, in string *object*, in string *attribute*, in string *value*, out string *retcode*)

Set an attribute for the given object.

Parameters:

namespace The namespace in which the value should be set.
object The object
attribute The name of the attribute
value The value of the attribute

Returns:

See **addAttribute()**, without 412.

void removeAttribute (in string *namespace*, in string *object*, in string *attribute*, in string *value*, out string *retcode*)

Remove an attribute from the given object.

Parameters:

namespace The namespace from which the value should be removed.

object The object

attribute The name of the attribute

value The value of the attribute

Returns:

see addAttribute(), without 409, 412, additional 400 Not found

void getAttribute (in string *object*, in string *attribute_name*, out set< AttributeValue > *result*, out string *retcode*)

Retrieve the values of the attribute of the object from all accessible namespaces.

The result set contains a local object identifier for each attribute value pair that allows the client to relate attributes to objects. Furthermore, each element of the result set contains the namespace name it is coming from.

Parameters:

object The object

attribute_name The name of the attribute

Returns:

200 Values retrieved

403 User not authorized

404 Object not found

406 Attribute name not found (Attribute does not fit to scheme)

503 Service Unavailable

void getAttributes (in string *object*, out set< AttributeValue > *result*, out string *retcode*)

Retrieve all the values of all the attributes (incl. public, user's private, internal) of the object.

Parameters:

object The object

Returns:

See above, without 406

void getAttributesForObjects (in set< string > *objects*, out set< AttributeValue > *result*, out string *retcode*)

Retrieve all the values of all the attributes (incl. public, user's private, internal) of all the given objects.

Parameters:

objects An array of LFNs for which to retrieve the attributes.

Returns:

See above, without 406.

void getObjects (in string *query*, out set< string > *objects*, out string *retcode*)

Execute the query and retrieve all object ids (ie. LFN) of the result set.

Queries are currently of the form "AttributeName1=Value1&&AttributeName2=Value2&&....".

This will be extended later.

Parameters:

query The query.

Returns:

200 Ok

403 User not authorized

420 Parse error

406 Attribute name not found (Attribute does not fit to scheme)

409 Value wrong type for operator

503 Service Unavailable

void getAttributesForQuery (in string *query*, out set< AttributeValue > *result*, out string *retcode*)

Execute the query and retrieve all attributes of the objects in the result set.

Parameters:

query The query.

Returns:

See above.

void getServiceDescription (out string *description*)

Give status information

Returns:

A string with service status information

2.3 Relations to other workpackages

2.3.1 WP9 - Portal

The Gridlab Portal, as GridLab's principle user interface, will contain a data management portlet which will present the Metadata Catalog's functionality to the user. We worked together with the Portal WP to begin integrating our web service interface with the portal.

2.3.2 WP12 - Messaging

The Messaging work package has developed a client library which supports the storage of message metadata within the Metadata Catalog. This library is communicating successfully via the web service interface.

2.4 Planned Improvements

This section lists improvements to the service that are planned for implementation before the final release.

2.4.1 Query language

Currently, the query language supports only conjunctions of predicates which contain the equality operator. We plan to enhance a query language to support parenthesis, other boolean query operators, and “greater than” and “smaller than” operators for predicates.

2.4.2 Schema Management

Currently, the metadata schema is implicitly defined by the way the namespaces are used. We plan to add an interface for explicit schema management.

3 Remote Partial File Access

In parallel to this deliverable we have also delivered D8.5. The document for that deliverable discusses the Remote Partial File Access mechanism. We also wrote a paper on the theoretical background which has been accepted for CCGrid 2004 [1].

We have integrated the remote partial file access with Amira to provide efficient visualization of remote data. The next associated step will be to integrate partial file access support with Triana.

4 Replica Catalog

The replica catalog consists of 3 components:

- a namespace for logical file names;
- a database of replica locations; and
- an interface to file movement to replicate files on demand.

4.1 Namespace for logical file names

In the last release, our interface to the replica provided two abstractions: logical files and collections of logical files. After speaking with users, we came to the conclusion that a more powerful abstraction was necessary. We eventually decided to use a model which is already in widespread use: directories and files. Following this approach, we changed our API to reflect its similarity with POSIX filesystems. The functions are now called: mkdir, rmdir, create, rm and ls. We do not guarantee POSIX semantics; instead, we roughly mimic the associated interface. As an aside, for performance reasons, it is difficult to guarantee consistency between replicas.

4.2 Database for replica locations

The replica location database has not changed significantly since the last release. The same three core functions remain:

- add location of file
- remove location of file
- list locations

In contrast to the last release, the logical file is now specified by a path instead of collection name + file name.

4.3 Interface to File Movement

The catalog provides several functions to create replicas of files at new locations. In the case where several physical locations are already known to the catalog, a suitable replica must be chosen for transfer to the new location. As the catalog has no notion of bandwidth or distance between hosts, we rely on the Adaptive Service from WP-7 to provide a ranking of the replicas by available bandwidth to the destination host.

The actual file transfer is provided by the Data Movement Service.

4.4 Integration with other Services

The earliest user of the Replica Catalog were the authors of the Resource Management Service. They use the catalog for registering input and output files as well as executables. The Replica Catalog also plays an important role for job migration.

In parallel, we began integration with the GAT. For both versions, the replica management adaptor was one of the earliest adaptors completed for the GAT.

In the last quarter, we began integration with the Visualization Service. If jobs register their output files or intermediate results, the visualization service can utilize logical files names instead of physical locations. This approach is much more user-friendly.

5 File movement

The file movement service's interface and behavior was previously implemented and shared as a prototype, as described in our deliverable 8.3. Based on the valuable feedback provided by other work packages (including WP2, WP4, WP5 and WP9) this interface was modified and extended, and has evolved into the form presented here as a first release of the data movement service.

5.1 ns_FileTask Struct Reference

xsd_int ns_FileTask::error_code

Numeric error code (200 - OK, 300 - service error, other - ftp error codes)

xsd_string ns_FileTask::error_string

Human-readable error string

xsd_long ns_FileTask::id

Unique (per service instance) identifier of the transfer

xsd_double ns_FileTask::progress_percentage

Percentage of completion for copy and move

xsd_int ns_FileTask::status

Operation status : 0 - New, 1 - Active, 2 - Pending, 3 - Failed, 4 - Done

xsd_int ns_FileTask::type

Operation type : 0 - Copy, 1 - Move, 2 - Delete

`xsd_string ns_FileTask::user_DN`

DN of the user that initiated the transfer

5.2 Function Documentation

`int ns_DATACopyFile (char * source_URL, char * dest_URL, int max_retries, int use_parallel, char ** response)`

Copies a single file.

This method blocks until the operation is completed (with success or failure)

Parameters:

source_URL the URL of the source file

dest_URL the URL of the destination

max_retries the maximum number of restarts to be tried before a complete failure (momentarily unused)

use_parallel true if the usage of parallel streams is desired

response return parameter

Returns:

a string containing a numeric error code followed by a human-readable description

`int ns_DATACopyFileDefaults (char * source_URL, char * dest_URL, char ** response)`

Copies a file using default parameters.

This method blocks until the operation is completed (with success or failure). The defaults are currently 0 for `max_retries` and 1 for `use_parallel`.

Parameters:

source_URL the URL of the source file

dest_URL the URL of the destination

response return parameter

See also:

[ns_DATACopyFile](#)

Returns:

a string containing a numeric error code followed by a human-readable description

`int ns_DATADeleteFileDefaults (char * in_URL, char ** response)`

Deletes a file.

This method blocks until the operation is completed (with success or failure).

Parameters:

in_URL the URL of the file

response return parameter

Returns:

a string containing a numeric error code followed by a human-readable description

int ns_DATAdeleteFileTask (xsd_long *id*, int * *response*)

Deletes information about a transfer.

The transfer should have been initiated with one of the init methods. This method stops the transfer if in progress. After delete is called, the information about the transfer will be erased and the id will be reused. This method should always be called after the transfer is completed so that the memory on the machine hosting the service can be freed.

Parameters:

id the transfer identifier

response the return parameter

Returns:

integer containing error code

int ns_DATAgetFileTask (xsd_long *id*, struct ns_FileTask * *response*)

Returns information about the status of a transfer.

The transfer should have been initiated with one of the init methods. The method returns immediately.

Parameters:

id the transfer identifier

response the return parameter

Returns:

a handle containing data on the transfer status

int ns_DATAinit_CopyFile (char * *source_URL*, char * *dest_URL*, int *max_retries*, int *use_parallel*, struct ns_FileTask * *response*)

Starts a non-blocking copy operation.

Non-blocking operations have similar parameters to the blocking versions. Every non-blocking operation is handled by a separate thread.

See also:

[ns_DATACopyFile](#)

Parameters:

response the return parameter

Returns:

a handle containing the transfer id and other data on the transfer status

int ns_DATAinit_DeleteFile (char * *source_URL*, struct ns_FileTask * *response*)

Starts a non-blocking delete operation.

See also:

[ns_DATAinit_CopyFile](#)

Parameters:

source_URL the URL of the file

response the return parameter

Returns:

a handle containing the transfer id and other data on the transfer status

int ns_DATAinit_MoveFile (char * *source_URL*, char * *dest_URL*, int *max_retries*, int *use_parallel*, struct ns_FileTask * *response*)

Starts a non-blocking move operation.

See also:

[ns_DATAMoveFile](#) , [ns_DATAinit_CopyFile](#)

Parameters:

response the return parameter

Returns:

a handle containing the transfer id and other data on the transfer status

int ns_DATAinit_TransferFile (int *operation*, char * *source_URL*, char * *dest_URL*, int *max_retries*, int *use_parallel*, struct ns_FileTask * *response*)

container method for non-blocking Copy, Move and Delete.

Parameters:

operation 0 for copy, 1 for move and 2 for delete

source_URL the URL of the source file

dest_URL the URL of the destination (ignored for delete)

max_retries the maximum number of restarts to be tried before a complete failure(momentarily unused)

use_parallel true if the usage of parallel streams is desired at copy (ignored for delete)

response the return parameter

See also:

[ns_DATATransferFile](#) , [ns_DATAinit_CopyFile](#)

Returns:

a handle containing the transfer id and other data on the transfer status

int ns_DATAMoveFile (char * *source_URL*, char * *dest_URL*, int *max_retries*, int *use_parallel*, char ** *response*)

Copies a file and then deletes the source.

This method blocks until the operation is completed (with success or failure).

Parameters:

source_URL the URL of the source file

dest_URL the URL of the destination

max_retries the maximum number of restarts to be tried before a complete failure(momentarily unused)

use_parallel true if the usage of parallel streams is desired at copy

response return parameter

Returns:

a string containing a numeric error code followed by a human-readable description

int ns_DATAMoveFileDefaults (*char * source_URL*, *char * dest_URL*, *char ** response*)

Moves a file using default parameters.

This method blocks until the operation is completed (with success or failure) The defaults are currently 0 for *max_retries* and 1 for *use_parallel*.

Parameters:

source_URL the URL of the source file

dest_URL the URL of the destination

response return parameter

See also:

[ns_DATAMoveFile](#)

Returns:

a string containing a numeric error code followed by a human-readable description

int ns_DATArestartFileTask (*xsd_long id*, *struct ns_FileTask * response*)

Restarts a stopped or otherwise interrupted copy or move operation.

The transfer should have been initiated with one of the init methods. If the operation was interrupted with *stopFileTask* or by a transient network error and the service was previously able to save a restart marker, the operation is resumed using this call. If the transfer was not interrupted (it failed from the beginning) or it was interrupted too early (no restart marker was saved) then the transfer cannot be resumed.

Parameters:

id the transfer identifier

response the return parameter

Returns:

a handle containing the transfer id and other data on the transfer status

int ns_DATAstopFileTask (*xsd_long id*, *struct ns_FileTask * response*)

Stops a copy or move operation (blocking).

The transfer should have been initiated with one of the init methods. This methods interrupts (if possible) the transfer. This is especially useful for large file transfers. The operation may be resumed using *restartFileTask*

Parameters:

id the transfer identifier

response the return parameter

Returns:

a handle containing data on the transfer status

int ns_DATATransferFile (*int operation*, *char * source_URL*, *char * dest_URL*, *int max_retries*, *int use_parallel*, *char ** response*)

Container method for Copy, Move, and Delete.

This method makes calls to Copy, Move, or Delete depending on the operation type. The other parameters are forwarded. This method blocks until the operation is completed (with success or failure).

Parameters:

operation 0 for copy, 1 for move and 2 for delete

source_URL the URL of the source file

dest_URL the URL of the destination (ignored for delete)

max_retries the maximum number of restarts to be tried before a complete failure (momentarily unused)

use_parallel true if the usage of parallel streams is desired at copy (ignored for delete)

response return parameter

See also:

[ns_DATACopyFile](#) , [ns_DATAMoveFile](#) , [ns_DATADeleteFileDefaults](#)

Returns:

a string containing a numeric error code followed by a human-readable description

int ns_DATAwaitFileTask (xsd_long *id*, struct ns_FileTask * *response*)

Blocks and waits for a transfer to complete.

The transfer should have been initiated with one of the init methods. This method waits for the transfer to end; or, if the transfer is already finished, it returns immediately.

Parameters:

id the transfer identifier

response the return parameter

Returns:

a handle containing data on the transfer status

int ns_getServiceDescription (void * *dump*, char ** *description*)

Returns a string description of the service.

Parameters:

dump unused

description return parameter

int ns_isAlive (int *dump*, int * *response*)

Test method.

Returns 1 (used to see if the service is running)

Parameters:

dump unused

response return parameter

6 File browsing

The file browsing service's interface and behavior was implemented as a prototype as described in our deliverable 8.4. Based on the valuable feedback provided by other work packages (including WP2 and WP4) this interface was modified and extended, and has evolved into the form presented here as a first release of the file browsing service.

6.1 Directory_entry Struct Reference

char* Directory_entry::datetime

last modification time (example: Apr 8 2003)

char* Directory_entry::group

file group

long Directory_entry::links

number of links

char* Directory_entry::name

file name

char* Directory_entry::owner

file owner

char* Directory_entry::permissions

textual permissions (example: drwx—)

long long Directory_entry::size

file size in bytes

6.2 Directory_entries Struct Reference

struct [Directory_entry](#)* Directory_entries::_ptr

pointer to directory entries

int Directory_entries::_size

vector size

6.3 List_response Struct Reference

char* List_response::response

error code (numeric) followed by human-readable text

char* List_response::retlist

the string containing the directory listing

6.4 List_structured_response Struct Reference

struct [Directory_entries](#) List_structured_response::entries

The structured directory listing.

char* List_structured_response::response

error code (numeric) followed by human-readable text

6.5 Size_response Struct Reference

char* Size_response::response

error code (numeric) followed by human-readable text

unsigned long Size_response::size

size in bytes

6.6 Time_response Struct Reference

char* Time_response::response

error code (numeric) followed by human-readable text

long Time_response::seconds

Modification time in seconds.

6.7 Function Documentation

int ns_DATAConnectedList (char * *in_URL*, int *verbose*, struct [List_response](#) * *resp*)

Lists the content of a directory.

This is the main function of the browsing service. When called for the first time, this method initiates a cached connection to the GridFTP server if possible. That connection will be used for all subsequent list operations until the StopCache method is called. Also, the service will not cache more than a fixed number of connections at one time.

Parameters:

in_URL the URL of the directory to be listed

verbose A boolean parameter. When set to 1 will result in detailed listing of the given directory (ls -l). When set to 0, the method will return just a list of entries in that directory.

resp the return parameter

Returns:

the unparsed directory listing and an error message

See also:

[ns_DATAStopCache](#)

int ns_DATAConnectedListStructured (char * *in_URL*, struct [List_structured_response](#) * *resp*)

Lists and parses the content of a directory.

This method provides a structured response to a verbose directory listing. The text returned by the server is parsed and separated in directory entries. A cached connection will be used or initiated for this operation.

Parameters:

in_URL the URL of the directory to be listed

resp the return parameter

Returns:

the structured directory listing and an error message

See also:

[ns_DATAConnectedList](#) , [ns_DATAStopCache](#)

int ns_DATAConnectedModTime (char * *in_URL*, struct [Time_response](#) * *resp*)

Returns the modification time of a directory entry.

Returns the time since the Epoch (00:00:00 UTC, January 1, 1970), measured in seconds. A cached connection will be used or initiated for this operation.

Parameters:

in_URL the URL of the directory entry

resp the return parameter

Returns:

the modification time and an error message

See also:

[ns_DATAConnectedList](#) , [ns_DATAStopCache](#)

int ns_DATAConnectedSize (char * *in_URL*, struct [Size_response](#) * *resp*)

Returns the size of a file.

Returns the file size in bytes. A cached connection will be used or initiated for this operation.

Parameters:

in_URL the URL of the directory entry

resp the return parameter

Returns:

the file size and an error message

See also:

[ns_DATAConnectedList](#) , [ns_DATAStopCache](#)

int ns_DATAStopCache (void * *dump*, char ** *response*)

Destroys the data structure that was used to cache the GridFTP connections.

This method deletes the client handle, cached for the user connecting to the service. This removes all cached connections for the user. The method should be used when the user does not need to make any other listing operations to free the memory.

Parameters:

dump not used

response the return parameter

Returns:

error code (numeric) followed by human-readable text

int ns_getServiceDescription (void * *dump*, char ** *description*)

Returns a string description of the service.

Parameters:

dump unused

description return parameter

int ns_isAlive (int *dump*, int * *response*)

Test method.

Returns 1 (used to see if the service is running)

Parameters:

dump unused

response return parameter

References

- [1] T. Schütt, A. Merzky, A. Hutanu, and F. Schintke. Remote partial file access using compact pattern descriptions. Accepted for CCGrid 2004. [6](#)