EGEE

EGEE gLite User's Guide

GLITE METADATA CATALOG – GENERIC

Document identifier:	**EGEE-TECH-573725-v1.0**
Date:	**March 15, 2005**
Activity:	**JRA1: Data Management**
Document status:	**DRAFT**
Document link:	**https://edms.cern.ch/document/573725**

Abstract: This is the user's guide to the gLite Metadata Catalog. The API reference is language-neutral. The language-specific guides with examples are separate. The guide is mainly for people to explain the concepts and usage.

Document Change Log

Issue	Date	Comment	Author
1.0	1.3.2005	Initial Version.	Text by Ricardo Rocha and Peter Kunszt

Document Change Record

Issue	Item	Reason for Change

CONTENTS

1. INTRODUCTION

Metadata is in general a notion of 'data about data'. There are many aspects of metadata, like descriptive metadata, provenance metadata, historical metadata, security metadata, etc. Due to this multi-faceted nature of metadata, it is very difficult to define an interface which will suit all possible applications.

However, we do identify three concepts which is can be often encountered when discussing metadata:

Collections In many applications data entries are grouped together into collections (or groups, datasets, classes, etc). The collections may have different semantics from application to application (having set or group semantics for example) but the few most basic operations (add and remove member) remain universal. The collections we define here are exclusive, i.e. if an item belongs to a collection, it may not belong to another collection as well. A collection cannot be member of another collection either, so we do not support the concept of subcollections either.

Schemas The metadata itself always comes with a schema. The schema describes the actual metadata attributes, gives them names and types and also relations between each other. Each collection has a schema, but the same schema may be shared by many collections. For each schema there is a default collection with the same name as the schema itself, acting as the catch-all collection for the given schema in operations where no collection is specified, or for entries that do not belong to any collection.

Access Control Lists Access to the metadata needs to be often controlled in a very fine grained way, sometimes metadata needs to be better secured than the data itself (for example the crypto keys to encrypted data, patient data to medical files..)

We base the interface describe in this document on experience by the usage of the Replica Metadata Catalog in the EU DataGrid and the LHC Computing Grid projects, on the definition and prototype implementation of the ARDA Metadata interface [1] and the HEP Metadata Group Use Cases document [3] (and references therein). In addition, there are a set of use cases and requirements coming from the High Energy Physics and Biomedical application domains which are recorded in the EGEE Project Technical Forum database [4].

1.1. BASIC CONCEPTS

The concepts to understand the interfaces in the Metadata service are explained here. We start with defining our terminology:

Entry/Item A record inside the catalog. An entry has:

- a key - a string that contains the entry by which the metadata will be referenced. In the Grid case this may be for example a GUID or an LFN. This key must be unique.
- a list of attributes - each may hold information (metadata) about the entry referenced by the key. These attributes are used when querying the metadata.

Attribute An element to be associated with a schema. An attribute has:

- a name - a string that identifies the attribute within the catalog.
- a value - represented/encoded as a string.
- a type - a string indicating what kind of information is contained in the value, that can be used by the backend to decide how to store the values.

Attributes must be unique in that they can have one single type only, even when belonging to different schemas (to allow consistent queries on more than one schema). For example, it is not allowed to have an attribute called "date" which is defined with integer type in one schema and with string type in another schema - the two definitions must match. If such a thing is necessary, give it a different, more descriptive name like "stringdate" and "intdate" (which is good practice anyway to avoid mistakes on the application side.)

Schema A group of different attributes. Each entry/item in the catalog can be associated with one schema. A schema has:

- a name, which is a string that must be unique within the catalog.
- an attributeList, which contains all the attributes that belong to the schema.

Permission A permission in gLite consists of

- a BasicPermission, which stores a username, groupname and permission numbers for the user, group members and others.
- a list of ACLs. The ACL in gLite consists of a principal name (which can be any string, so either username or group name) and a list of access control bits for read, write, execute, list, remove, permission, getMetadata and setMetadata.

Depending on the context of the operation, the Permission semantics may be different.

The current gLite Metadata Catalog functionality is spread over four interfaces. These are described in the quickstart and reference sections in detail.

MetadataBase Two sets of operations are offered through this interface:

- Querying and setting values of attributes for individual entries/items in the catalog.
- Generic queries returning entry/item identifiers.

MetadataCatalog The necessary operations for managing entries in the catalog are available through this interface. This includes functionality for creating and deleting items. Also Collection-specific operations are in this interface.

MetadataSchema The necessary operations for handling schemas inside the catalog.

FASBase The set and get permission methods are inherited in MetadataBase from this interface.

An implementation of a Metadata Catalog can choose not to have all the functionality defined in the three interfaces. A concrete example is a File Catalog that also wants to offer file metadata. Functionality for managing entries/items in the catalog is already provided as part of the File Catalog, so the Metadata-Catalog interface described above is not needed. The MetadataBase interface would be enough to have POSIX xattrs functionality, and it can be extended with the MetadataSchema interface if desired.

In this guide we assume that a standalone metadata service is used, also implementing the MetadataCatalog interface, however most of the guide applies to services that implement only the MetadataBase and MetadataSchema interfaces, like the Fireman catalog.

1.2. Interactions with other Services

The MetadataBase interface is also implemented by the Fireman catalog, thus giving the ability to add metadata attributes to the files in the file catalog. Other than that, the Metadata Catalog is a standalone service that others may depend on but it does not depend on any other grid service (except for the usual security mechanism dependencies, e.g. VOMS).

For an explanation of how the metadata catalog fits into the whole set of data management components in gLite, see also the Overview of gLite Data Management User's Guide [2].

2. Reference Guide

In this section we go through the methods in detail and describe their semantics. The Interface Inheritance diagram 1 shows the interfaces described here.

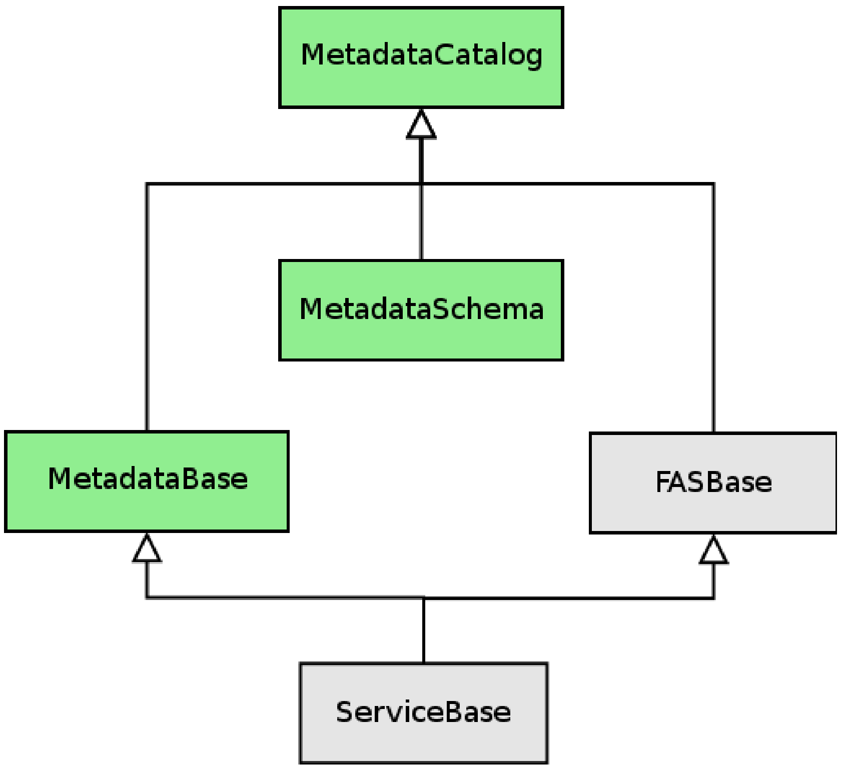

Figure 1: The Metadata Interface inheritance. The Base classes include generic methods, while the MetadataSchema and MetadataCatalog classes contain the metadata-specific operations.

2.1. Base Operations

This section defines the `MetadataBase` interface. The Operations in this interface are the following:

`setAttributes` Sets the values for a group of attributes of items/entries.

`getAttributes` Gets a group of attributes of a given item/entry.

`clearAttributes` Clears the values for a group of attributes.

listAttributes List all attributes associated with an entry.

query Performs a generic query on the catalog, returning the corresponding item's identifiers.

2.1.1. SETATTRIBUTES

```
int setAttributes(MDQuery query, Attribute[] attributes)
```

This method sets the values for a group of attributes of items/entries. The `MDQuery` object is a structure as defined in section **??**. It contains information on what entries should be affected by this operation. The attributes parameter is a list (array) of `Attribute` objects (see section 2.6.1., each containing the necessary information to set the new values (`name/type/value` triplet). The `name` and `value` parameters are mandatory in each `Attribute`.

An implementation can choose to offer pure POSIX xattrs semantics, allowing on the fly creation of attributes when they do not exist yet – note that this change is done on a per-schema basis, so all entries belonging to the entries sharing the same schema will be affected. The `type` field in each `Attribute` object should be filled in this case. Or it may not support this feature, and the `type` field can be left null.

Return Value: The number of entries in the catalog that were affected by the operation.

Errors

AuthorizationException No access right to update values of attributes.

NotExistsException One of the attributes specified does not exist.

InvalidArgumentException One of the `name/type` pairs given for an attribute is invalid in the case of on-the-fly creation of attributes, where the `type` given is unsupported by the backend. Or one of the attribute's values is invalid while trying to update values. Or the `MDQuery` object given is invalid.

InternalException Any other error on the server side (i.e. database down).

2.1.2. GETATTRIBUTES

```
Attribute[] getAttributes(String item, String[] attributeNames)
```

This method gets a group of attributes of a given item/entry. The item is the unique identifier of items/entries in the catalog. It can be a GUID, an LFN, or any other kind of string identifier. The attributeNames is a list of the names of the attributes inside the catalog that should be returned for the given item.

Return Value: A list of `Attribute` objects containing the requested values. Each `Attribute` object MUST contain the `name` and `value` filled in, and MAY contain the type filled in. Returns an empty array if there is no attribute in the catalog with any of the given names.

Errors

`AuthorizationException` No access right to access attributes for this item.

`NotExistsException` The item/entry or one of the attributes specified does not exist.

`InternalException` Any other error on the server side (i.e. database down).

2.1.3. CLEARATTRIBUTES

```
void clearAttributes(MDQuery query, String[] attributeNames)
```

This method clears the values for a group of attributes. The query object is a structure as defined in the `MDQuery` object in section 2.6.2.. It contains information on what entries should be affected by this operation. The attributeNames is a list of the names of the attributes inside the catalog.

Return Value: The number of entries in the catalog that were affected by the operation.

Errors

`AuthorizationException` No access right to update values of attributes.

`InvalidArgumentException` The MDQuery object is invalid.

`NotExistsException` One of the attributes specified does not exist.

`InternalException` Any other error on the server side (i.e. database down).

2.1.4. LISTATTRIBUTES

```
Attribute[] listAttributes(String item)
```

This method lists all attributes associated with an entry. The item passed in the request is the unique identifier of the entry in the catalog.

Although the semantics of the method involve returning the names and types of the attributes inside the catalog associated with a given entry, an implementation MAY decide to include the values of the attributes inside each `Attribute` object (see section 2.6.1.) in the returned list. All attributes associated with the entry MUST be returned - including the ones with no value set.

Return Value: An array of `Attribute` objects where the name and the type MUST be filled. Optionally, the value MAY also be filled, allowing a behaviour similar to a single-shot getAllAttributes operation - that does not exist in this interface. Returns null if the item/entry has no attributes.

Errors

`AuthorizationException` No access right to access attributes for this entry/item.

`NotExistsException` The item/entry specified does not exist.

`InternalException` Any other error on the server side (i.e. database down).

2.1.5. QUERY

```
String[] query(MDQuery query, long time, int limit, int offset);
```

This method performs a generic query on the catalog, returning the corresponding item's identifiers. The query is given by a `MDQuery` object which is described in Section 2.6.2.. The limit is the number of records to be returned in a single request. The offset is the number of records that should be ignored from the beginning of the whole resultset of the query.

The method returns the identifiers of all items corresponding to the given query. These identifiers are strings, each being for exaxmple a GUID, LFN, or any other unique string identifier. The number of these items returned in each call to the service can be:

Limit provided in the query if the number of items corresponding to the query minus the offset is bigger than this limit.

The number of items left when the total number of items corresponding to the query minus the offset is smaller than the limit provided.

The service may also enforce its own limits, to be retrieved by the `getServiceMetadata` call (see Section **??** asking for the query limit parameter.

The `MDQuery` object MUST contain the scope of the query, which in our case is the name of the schema or the collection which the query should operate on.

Return Value: The string identifiers of all items corresponding to the given query. An empty array if there are no items to be returned. This may happen when the offset is above the total number of items corresponding to the query, and is the condition for checking the end of the iteration through the results.

Errors

`AuthorizationException` No access right to list the entries.

`InvalidArgumentException` The query string given is invalid, or the query type is invalid, or the limit given is invalid (below 0 or above the system limit), or the offset is invalid (below 0).

`InternalException` Any other error on the server side (i.e. database down).

2.2. SCHEMA OPERATIONS

This section defines the `MetadataSchema` interface. The Operations in this interface are the following:

`createSchema` Creates a new schema in the catalog.

`addSchemaAttributes` Adds new attributes to an existing schema.

`removeSchemaAttributes` Removes attributes from an existing schema.

`renameSchemaAttributes` Renames an attribute in an existing schema.

`describeSchema` Get the full description of an existing schema in the catalog.

`dropSchema` Drops an existing schema from the catalog.

`listSchema` Lists all existing schemas in the catalog.

2.2.1. CREATESCHEMA

```
void createSchema(String schemaName, Attribute[] attributes)
```

This method creates a new schema in the catalog. A schema is a group of attributes. A schema has a name so that this group can be easily refered to. Each of this attributes will have its own name, type and value. (See Section 2.6.1. for a description of the `Attribute` object. The attributes parameter is a list of `Attribute` objects. Each object MUST have the `name` and `type` fields filled in. The `value` field in the MAY be used for having default values. Schemas MUST be unique within the catalog.

An item/entry inside the catalog is associated with a schema. When a client queries or updates the attributes of an item/entry, these must be contained in the associated schema.

Errors

`AuthorizationException` No access for client to create new schemas.

`ExistsException` A schema with the same name already exists in the catalog, or there is more than one attribute with the same name in the list of attributes given.

`InvalidArgumentException` One of the attributes given is invalid. It may be due to an invalid attribute name (i.e. empty), or the type given not being supported in the catalog.

`InternalException` Any other error on the server side (i.e. database down).

2.2.2. ADDSCHEMAATTRIBUTES

```
void addSchemaAttributes(String schemaName, Attribute[] attributes)
```

This method adds new attributes to an existing schema. Attributes MUST be unique within the whole catalog, so there is no way of creating an attribute with a different type than an attribute with the same name in another schema.

The schemaName is the name of the schema where attributes should be added. The attributes parameter is simply a list of `Attribute` objects where the `name` and `type` field MUST be filled in, and the `value` field MAY be used by the implementation to have default values for attributes in the schema.

Errors

`AuthorizationException` No access for client to add attributes to the schema.

`NotExistsException` The given schema does not exist in the catalog.

`ExistsException` There is already one attribute in the schema with the same name as one of the attributes in the list given. Or there is more than one attribute with the same name in the list.

`InvalidArgumentException` One of the attributes given is invalid. It may be due to an invalid attribute name (i.e. empty), or the type given is not supported in the catalog.

`InternalException` Any other error on the server side (i.e. database down).

2.2.3. REMOVESCHEMAATTRIBUTES

```
void removeSchemaAttributes(String schemaName, String[] attributeNames)
```

This method removes attributes from an existing schema. The semantics of the removal process are up to the implementation to decide. It can remove the attribute from the schema even if there are items/entries in the catalog with values set for it. Or it can decide to remove only if there are no items/entries actually using this attribute at the time – ie. none has the value set. It can also decide to remove an attribute from a schema when no item/entry in the catalog is associated with that same schema at the time of the request.

The schemaName is the name of the schema where the attributes will be removed. The attributeNames is simply a list of the names of the attributes that should be removed from the schema.

Errors

`AuthorizationException` No access for removing attributes from the schema.

`NotExistsException` The given schema or one of the attributes given does not exist in the catalog.

`InternalException` Any other error on the server side (i.e. database down).

2.2.4. RENAMESCHEMAATTRIBUTES

```
void renameSchemaAttribute(String schemaName, String attributeName, String newName)
```

This method renames an attribute in an existing schema. The `schemaName` is the name of the schema where the attribute will be renamed. The `attributeName` is the current name and the `newName` is the new name of the attribute.

Errors

`AuthorizationException` No access for changing attributes in the schema.

`NotExistsException` The given schema or one of the attributes given does not exist in the catalog.

`InternalException` Any other error on the server side (i.e. database down).

2.2.5. DROPSCHEMA

```
void dropSchema(String schemaName)
```

This method drops an existing schema from the catalog. The semantics of this operation are up to the implementation. It may be that a schema can only be dropped if no items/entries in the catalog are associated with it at the time of the request. Or it may be that schemas are dropped even when there are items/entries associated with them. This would mean the existing metadata would be lost.

The schemaName is the name of the schema to be dropped from the catalog.

Errors

`AuthorizationException` No access for client to drop the schema.

`NotExistsException` The given schema does not exist in the catalog.

`InternalException` Any other error on the server side (i.e. database down).

2.2.6. LISTSCHEMA

```
String[] listSchemas()
```

This method lists all existing schemas in the catalog. The list of strings returned contains only the names of the schemas, not the whole description. To get the details on each of the schemas in the catalog, an additional request should be made to `describeSchema`, where all the attributes and their descriptions will be returned.

Errors

`AuthorizationException` No access to list schemas for the client.

`InternalException` Any other error on the server side (i.e. database down).

2.2.7. DESCRIBESCHEMA

```
Attribute[] describeSchema(String schemaName)
```

This method gets the full description of an existing schema in the catalog. A schema is described by the attributes it contains. Each of these attributes is represented as an `Attribute` object, so the description of a schema is simply a list of `Attribute` objects. The schemaName is the name of the schema to return the description of.

Return Value: A list of `Attribute` objects fully describing all elements of the schema. Each of these objects MUST have the `name` and `type` fields filled in. The `value` type MAY also be used, but only when the implementation wants to get default values for attributes in schemas. Returns an empty list when the schema contains no attributes.

Errors

`AuthorizationException` No access for client to get the schema description.

`NotExistsException` The schema requested does not exist in the catalog.

`InternalException` Any other error on the server side (i.e. database down).

2.3. STANDALONE METADATA CATALOG OPERATIONS

This section defines the `MetadataCatalog` interface. The Operations in this interface are the following:

`createEntry` Creates new items/entries in the catalog.

`removeEntry` Removes items/entries from the catalog.

`createCollection` Creates a new collection in the catalog.

`removeCollection` Removes collection from the catalog.

`assignCollectionEntry` Adds entries to an existing collection.

`getEntryCollection` Retrieve the collection name to which the entry belongs.

`getCollectionSchema` Retrieve the schema name associated to a collection.

`listCollection` List all collections for a given schema.

2.3.1. CREATEENTRY

```
void createEntries(String collection, String[] items)
void createEntry(String collection, String item, Attribute[] attributes);
```

These methods create new items/entries in the catalog. Each item/entry is associated with an existing collection, which in turn has a schema. If the schema name is given for the `collection` argument, then the default collection for this schema is chosen.

The first form of the method just creates the items associated with the collection or schema. This method can be used to create several items in one go. The second form of the method just takes a single entry but it is possible to set its attributes at creation time.

Errors

`AuthorizationException` No access for creating new entries/items.

`ExistsException` The entry/item already exists in the catalog.

`NotExistsException` The collection/schema does not exist in the catalog.

`InvalidArgumentException` The given string identifier for the new item/entry is invalid. Or if both schema and collection are specified, the schema is not the associated schema of the collection. Or one of the name/type pairs given for an attribute is invalid. Or one of the attribute's values is invalid while trying to set values.

`InternalException` Any other error on the server side (i.e. database down).

```
    void removeEntries(String[] entries)
    void removeEntry(MDQuery query)
```

2.3.2. REMOVEENTRY

These methods remove existing items/entries from the catalog. The concrete semantics of this operation are left to the implementation. It may be that all entries should be deleted even if attributes are set, or that entries should only be deleted if no associated attributes have values set. The first form simply takes a list of entry names, the second form takes an `MDQuery` object, removing all the entries that are returned by the given query. If the entries are part of a collection, they will be removed from the collection as well.

Errors

`AuthorizationException` No access for deleting entries/items.

`InternalException` Any other error on the server side (i.e. database down).

`InvalidArgumentException` The query object is invalid.

2.3.3. CREATECOLLECTION

```
    void createCollection(String name, String schema)
```

This method creates a new collection in the catalog. The collection will be associated with the given schema. Each item/entry later associated with this collection will have all attributes defined in the schema.

Errors

`AuthorizationException` Client is not authorized to add collections.

`ExistsException` There is already a collection with a given name in the catalog.

`InvalidArgumentException` The given collection name is invalid – it contains invalid characters, etc.

`NotExistsException` There is no schema in the catalog with the given name.

2.3.4. REMOVECOLLECTION

```
    void removeCollection(String name, Boolean removeContent)
```

This method removes the collection from the catalog. The collection with the given name is removed. If the `removeContent` flag is set, all the members of the collection will be removed from the catalog as well. Otherwise the collection members will be assigned to the default collection of the schema (its name is just the schema name).

Errors

`AuthorizationException` The client is not allowed to remove the collection from the catalog.

`NotExistsException` The given collection does not exist in the catalog.

2.3.5. ASSIGNCOLLECTIONENTRY

```
void assignCollectionEntry(String collection, String[] entry)
```

This method reassigns entries from one collection to another existing collection with the same schema. If the collection name is the schema name, the entries are placed into the default collection of the given schema.

Errors

`AuthorizationException` Client is not authorized to add/remove entries to/from the collection.

`ExistsException` One of the entries is already member of the collection.

`InvalidArgumentException` The given collection schema and some of the entry schemas don't match.

`NotExistsException` There is no such collection or entry in the catalog.

2.3.6. GETENTRYCOLLECTION

```
String getEntryCollection(String entry)
```

This method retrievs the collection to which an entry belongs. If the entry is not part of a collection, the schema name is returned, which is the default collection for all entries.

Errors

`AuthorizationException` Client is not authorized to read the entry.

`NotExistsException` There is no such entry in the catalog.

2.3.7. GETCOLLECTIONSCHEMA

```
String getCollectionSchema(String collection)
```

This method retrievs the schema name which is associated to the given collection.

Errors

`AuthorizationException` Client is not authorized to read the collection.

`NotExistsException` There is no such collection in the catalog.

2.3.8. LISTCOLLECTION

```
String[] listCollection(String schema)
```

This method retrievs all the collections that are associated with the given schema.

Errors

`NotExistsException` There is no such schema in the catalog.

2.4. PERMISSIONS

This section defines the `FASBase` interface. The Operations in this interface are the following:

`setPermission` Sets full set of permissions BasicPermission,ACL for a given item.

`getPermission` Retrieves all set permissions for given items.

`checkPermission` Checks if the current user has the required permission bits on the specified items.

2.4.1. SETPERMISSION

```
void setPermission(PermissionEntry[] permissions)
```

This method sets the full set of permissions BasicPermission,ACL for a given item, replacing any previous permissions. The `PermissionEntry` object is described in Section 2.6.3..

Errors

`AuthorizationException` No access right to update the permissions.

`NotExistsException` The item does not exist.

`InvalidArgumentException` Some part of the argument is invalid.

2.4.2. GETPERMISSION

```
PermissionEntry[] getPermission(String[] items)
```

This method retrieves all permissions for the given items. The `PermissionEntry` object is described in Section 2.6.3..

Errors

`AuthorizationException` No access right to get the permission information.

`NotExistsException` The item does not exist.

`InvalidArgumentException` Some part of the argument is invalid.

2.4.3. CHECKPERMISSION

```
void checkPermission(String[] items, Perm perm)
```

This method checks if the current user has the required permission bits given by the `Perm` object on the specified items and returns with an AuthorizationException if for some items this is not the case.

Errors

`AuthorizationException` The permission check failed - some entries cannot be accessed with the given Perm.

`NotExistsException` An item does not exist

2.5. SERVICE OPERATIONS

This section defines the `ServiceBase` interface. The Operations in this interface are the following:

`getVersion` Retrieve the implementation version.

`getInterfaceVersion` Retrieves the service interface version being implemented.

`getSchemaVersion` Retrieve the version of the schema being used.

`getServiceMetadata` Retrieve the value of a given key associated with this service.

2.5.1. GETVERSION

```
String getVersion()
```

Return the server implementation version as a string.

2.5.2. GETINTERFACEVERSION

```
String getInterfaceVersion()
```

Return the interface version as a string.

```
String getSchemaVersion()
```

2.5.3. GETSCHEMAVERSION

Return the schema version as a string.

2.5.4. GETSERVICEMETADATA

```
String getServiceMetadata(String key)
```

Service metadata query. The `key` parameter has to be specified. The method will return the requested parameter or an empty string if it does not exist.

2.6. AUXILIARY OBJECTS

In this section we define all the auxiliary objects that are being used in the interface methods described above.

2.6.1. ATTRIBUTE

```
Attribute {

    String name
    String value
    String type
}
```

The `Attribute` object contains a name, a value and a type. It is the object with which items/entries in the Metadata Catalog can be associated.

An attribute in the Metadata Catalog is unique in the sense that the combination name/type MUST be unique. An attribute MUST have only one type, even when associated with more than one schema.

All values stored inside an Attribute are encoded as strings, even if the backend is storing them using a different type. The type of an Attribute is a hint on the content of this string, so that the backend can optimize storage.

2.6.2. MDQUERY

A query on the metadata catalog is represented using the `MDQuery` object. The object contains the name of the schema and/or collection(s) on which the query needs to be run. Either one of the two HAS TO be given to specify the scope of the query. If more than one collection is given they have to share the same schema. If the schema name is given for the collection, the default catch-all collection for the schema is chosen.

```
MDQuery {

    String[] collection
    MDSubQuery[] query
}


MDSubQuery {
    MDConstraint[] constraint
    MDSubQuery[] query
}


MDConstraint {

    String name
    Enumeration operation
    String value
}
```

The `MDConstraint` object describes a constraint on the value of a given object. The `operation` is a well-defined set of operations, enumerated as an integer. A full query can be assembled using the `MDSubQuery` objects, which may contain a list of constraints (which will be ANDed together) and a list of further sub-queries (which are ORed). Any of the two lists in `MDSubQuery` may be empty.

Example:

```
(age < 30 AND age > 18) OR (maritalstatus = "married")

MDQuery {
  collection = "car-drivers"
  query      = {q1, q2}
}

MDSubQuery q1 {
  constraint = {c1, c2}
}

MDSubQuery q2 {
  constraint = {c3}
}

Constraint c1 {
  name       = age
  operation  = LT
  value      = 30
}

Constraint c2 {
  name       = age
  operation  = GT
  value      = 18
}

Constraint c3 {
  name       = maritalstatus
  operation  = EQ
  value      = "married"
}
```

This may seem complicated but is quite straight forward and makes sure that the syntax is correct. gLite intends to provide client-side utilities that convert SQL or BNL strings into MDQuery objects.

2.6.3. PermissionEntry

```
PermissionEntry {

    Permission permission
    String item
}

Permission {

    ACLEntry[] acl
}

ACLEntry {

    Perm principalPerm
    String principal
}

Perm {

    Boolean permission
    Boolean remove
    Boolean read
    Boolean write
    Boolean list
    Boolean execute
    Boolean getMetadata
    Boolean setMetadata
}
```

PermissionEntry contains a single item permission. The item is identified by a string, being the catalog entry item and the permission is a `Permission` object. The `Permission` object is composed of a list of

`ACLEntry` objects, which in turn is a named principal (like a DN or a group name) and the associated permission bits, given by the `Perm` object. The single permission bits have different meanings for each of the metadata operations.

3. KNOWN ISSUES AND CAVEATS

One of the main open conceptual questions is how collections should behave. The current model described in this document is the first-order simplistic model where each item in the metadata catalog may belong to at most one collection and collections may not have subcollections. Both of these restrictions make the life of the implementor easier.

The first constraint is due to the fact that the membership in a collection is entirely an attribute of the metadata catalog entry in this model. If the entry can belong to several collections, this has to be represented separately, adding to the complexity of the metadata catalog schema.

Also, if collections need the additional flexibility of being able to contain one another, then this needs some additional representation, and all the issues with hierarchy will have to be dealt with (depth, loops, etc). It is an open question whether this added complexity is worth the benefits of the added functionality, also with respect to querying.

Another question is about schema evolution. How are changes to be dealt with? One of the suggestions is to have a timestamp-based rule of keeping track of all chage, but it is unclear how this maps to the schema concept. If an attribute is renamed, for example, how is this information to be kept using a time-stamp based approach?

REFERENCES

[1] N. Santos B. Koblitz. A Proposal for a Metadata Interface. Technical Note, January 2005. http://agenda.cern.ch/askArchive.php?base=agenda&categ=a05664&id=a05664s1t1/document.

[2] JRA1 Data Management Cluster. *Overview of gLite Data Management*. EGEE, March 2005. https://edms.cern.ch/document/570643.

[3] Steven Hanlon et. al. Unlucky for Some: The thirteen core use cases for HEP metadata. Technical Note, December 2004. http://www.gridpp.ac.uk/datamanagement/metadata/SubGroups/UseCases/CoreUseCases_v10.pdf.

[4] EGEE Project Technical Forum Requirements Database. https://savannah.cern.ch/support/?group=egeeptf.