

AMGA USER'S AND ADMINISTRATOR'S MANUAL

B. Koblitz, N. Santos

January 24, 2006

Abstract

This is the manual for users and administrators of AMGA. It intends to give an overview on the installation of the client and server packages as well as the client-api packages. Examples of the usage of the command line clients and the client APIs are given.

1 Overview

AMGA is a metadata service for the Grid. In a more general way this is a database access service for Grid applications which allows user jobs running on the Grid to access databases by providing a Grid style authentication as well as an opaque layer which hides the differences of the different underlying database systems from the user. To achieve this, AMGA is a service sitting between the RDBMS and the user's client application.

In addition to this database translation layer, AMGA intends to solve another problem database services face on the Grid which is latencies. AMGA intends to provide a replication layer which makes databases locally available to user jobs and replicate the changes between the different participating databases. A simple implementation based on PostgreSQL asynchronous replication is already working.

All examples given in this manual as well as additional documentation in particular the reference manuals of all APIs are given on the projects home page at <http://project-arda-dev.web.cern.ch/project-arda-dev/metadata/>

2 Installation

The server and the clients (C++, Java and Python) are provided as RPMs packages. RPMs are currently only supported for CERN Scientific Linux or RedHat Enterprise Linux. For users running other operating systems (including Windows and other unix flavours), the Java client is also provided as a platform-independent package (a tar ball), including the Java API, a command line client, some examples and the documentation. You can find the packages in the [download directory](#). Download the latest version. As of writing this it is 1.1.0.

You can install the `glite.amga.client` and the `python` and `java api` packages independently of the server package, however the server packages depend on the client package.

2.1 Client installation

To install the command line client and C++ api, you will only need to download the `glite-amga-cli-1.1.0.i386.rpm` itself. By default the package will be installed into `/opt/glite`, so you will need to have write permission for this directory. Install the package via

```
rpm -i glite-amga-cli-1.1.0.i386.rpm
```

Copy the `/opt/glite/etc/mdclient.config` client configuration file into the directory from which you intend to work or into `~/.mdclient.config` and customize it according to the instructions in **Configuration of the C++ and Java command line clients**(p. 3).

If the amga client was built against the editline library instead of the standard unix readline library, you will need to also install that RPM.

To install the RPM with Java API, download and install the file named `glite-amga-api-java-0.X.Y.rpm`. This RPM is architecture-independent and will install in any Unix platform that supports RPM

packages. This RPM contains only the Java API (no command line client or documentation). It installs the file `glite-amga-api-java.jar` in `/opt/glite/share/java/`. To use the Java API, this file must be included on the classpath.

The tar ball with the Java Client API does not need any special installation procedure, apart from unpacking it. It provides both the documentation, the Java API, and two command line utilities similar to the ones installed by the C++ RPM, with the advantage of running in any platform with a Java virtual machine implementation (must be at least Java 1.4 compliant). These command line tools are:

- **mdjavaclient** - an interactive command line shell with the server.
- **mdjavacli** - an utility to submit a single command to the server. For use in shell scripts.

The tar ball includes four scripts to start these command line tools: `mdjavaclient.sh/mdjavaclient.sh` for Linux and `mdjavacli.sh/mdjavacli.sh` for Windows. The interactive command line client can also be started directly by executing the class `arda.md.javaclient.ConsoleClient`.

There is also a Python Client API module available as an RPM package (`glite.amga.api-python-1.1.0-1.noarch`) which will install the modules under `/opt/glite/python2.2/site-packages/amga`. Alternatively you can use the Python2.3 RPM which works nicely on Debian Sarge. If you prefer a custom installation you can download the source tarball `glite.amga.api-python-1.1.0.tar.gz` and install it via the Python installation mechanism (if you don't know how this works, `./setup.py -help` should get you started).

2.2 Server installation

The AMGA server depends on the C++ client package as well as some external dependencies which are provided by the gLite team for apt-get based installation. See <http://glite.web.cern.ch/glite/packages/APT.asp>

The following packages are necessary to use AMGA and must be installed first: UnixODBC, libxml2 and Boost-lib, UnixODBC and libxml2 are standard SLC3 packages and can be also found in the usual CERN repositories. You can get the packages via

```
apt-get install unixODBC
apt-get install libxml2
apt-get install boost
```

If AMGA was compiled against the Globus environment and not the SLC3 environment in order to use the Globus versions of several SLC3 system libraries like openssl, then you will also need to install the Globus RPM. The gLite project currently only provides versions of AMGA which depend on Globus.

You also need a database and the appropriate ODBC driver. AMGA currently supports 4 different database backends via ODBC drivers. You will need to get at least one of the currently supported 4 database backends installed, including their ODBC driver. You have the choice among PostgreSQL, MySQL, Oracle and SQLite. The default is PostgreSQL and this database should be set-up correctly when installing the AMGA RPM if you have PostgreSQL and its ODBC driver installed:

```
apt-get install postgresql
apt-get install postgres-odbc
```

If you want to use a different database or you want to have a different setup of the ODBC driver, have a look at the more detailed instruction in the **Installation from Source**(p.18) .

You will need to get at least one of these 4 database backends in order to use the service. ODBC should have been installed correctly by the AMGA rpm, if you want to use a standalone installation using PostgreSQL. If you don't know about ODBC, some more hints can be found in the **Installation from Source**(p.18) or on a general page on ODBC like <http://www.unixodbc.org/doc/User-Manual/> the Unix ODBC User Manual .

Now you should install the server

```
rpm -i glite-amga-server-1.1.0.i386.rpm
```

If you want to use the PostgreSQL default installation, you can find the `init-arda-psql.db.sh` script in the `download` directory which should create a database user and set up the database access and initial tables automatically for PostgreSQL. This will only work on an SLC3 default installation.

If you want to do the setup manually, you first have to create a DB user, make sure he can connect via a TCP/IP connection (even locally this is required since ODBC does not work via a Unix Domain socket) and then set up an ODBC data source. Now you can initialise the database using the `createInitial.sql` (all DBs apart from MySQL) or `createInitialMySQL.sql` which you will find in `/opt/glite/share/doc/glite-amga-server-1.1.0/` after the installation of the server RPM:

```
For PostgreSQL:
psql -User db < createInitial.sql
and if you want to have ACLs on individual entries you need some stored procedures:
createlang plpgsql
psql -User db < proceduresPSQL.sql
For SQLite:
sqlite3 dbfile.db < createInitial.sql
or Oracle:
sqlplus username/pw@connectionID < createInitial.sql
or for MySQL:
mysql -uuser db < createInitialMySQL.sql
```

Finally activate this data source in the `mdserver.config` in `/opt/glite/etc/` file as described in **Configuring the AMGA Server**(p.13) .

You should now be able to start the service using

```
/etc/init.d/mdservice start
```

If you want to change the configuration of the metadata server, you should go on and read **Configuring the AMGA Server**(p.13) .

3 Configuration of the C++ and Java command line clients

The AMGA C++ command line clients as well as all other executables using the `MDClient` class of the C++ client package need to read an `mdclient.config` file for their configuration. This configuration file is searched first in the current directory, then as `$HOME/.mdclient.config` and finally as `/etc/mdclient.config`. Only the port and server of the configuration file can be overridden on the command line of the clients:

```
mdclient [-p port] [hostname]
```

The following is an example configuration file:

```
# Connection options
Host = localhost
Port = 8822

# User settings
Login = kobnitz
PermissionMask = rw-
GroupMask = r--
Home = /
#Name=

# Security options
UseSSL = require
#AuthenticateWithCertificate = 1
#CertFile=/home/kobnitz/.globus/usercert.pem
#KeyFile=/home/kobnitz/.globus/userkey.pem
UseGridProxy = 1
#Password = secret
#VerifyServerCert = 1
# If server certificates are verified, local certificates need to be loaded:
TrustedCertDir = /etc/grid-security/certificates
# RequireDataEncryption = 0
```

The following options are supported:

- **Host:** The name of the host to connect to. This option can be overridden on the command line of mdclient. (Default: localhost)
- **Port:** Port of the mdserver to connect to. Can be overridden on the command line of mdclient using the -p option. (Default: 8822)
- **Login:** The login name of the user on the AMGA server. All entries created in the catalogue will have this owner. This is also the user which you need to authenticate to the AMGA server if authentication is enabled. (Default: NULL which gives the default role when authenticating with a VO certificate)
- **PermissionMask:** A 3 character string giving the owner permissions of newly created entries in the metadata catalogue.
- **GroupMask:** A 3 character string giving the group permissions of newly created entries in the metadata catalogue.
- **Home:** The home-directory. The default is "/".
- **Name:** Can be used to give a full name to the server, comparable to the comment in an /etc/passwd file. Currently only used for information in the server.
- **UseSSL:** Possible values are no, try, require (synonyme is yes). Default is no. Needed for any authentication using certificates (also proxy certificate). You want this if you intend to use passwords which are not sent in plain text. If you use SSL the entire session will be encrypted. Some servers may require you to use SSL to connect. If you want to be sure that SSL is always used you need to set this to require or yes.
- **AuthenticateWithCertificate:** Set this to 1 to enable certificate based authentication, also grid-proxy certificates. You will need to either enable normal certificates via a Cert-File, KeyFile option pair, or use a grid proxy certificate via the UseGridProxy option. If you specify both, then the grid proxy gets precedence. This option does not work if you have not enabled SSL via UseSSL! In fact you will try to authenticate as the user named in Login, so you need to have this field set as well. However, the Password field is ignored unless certificate based authentication fails in which case the password is tried.
- **CertFile:** Path to your x509 certificate in .pem format. For this option to have any effect, you need also options UseSSL and AuthenticateWithCertificate enabled and UseGridProxy disabled!
- **KeyFile:** Path to your x509 private key in .pem format. If the key is encrypted you will be asked for the passphrase on client startup. For this option to have any effect, you need also options UseSSL and AuthenticateWithCertificate enabled and UseGridProxy disabled!
- **UseGridProxy:** Tries to use the a grid proxy certificate in /tmp/x509up_u[user-id].
- **Password:** If a password is given, password based authentication will be tried. You should want to use an SSL connection with this. This is *_not_* the password for your private key. For that you will always be prompted on the command line. For a discussion on how to escape special characters (^#=#) see **Configuring the AMGA Server**(p. 13) .
- **VerifyServerCert:** Verifies the server certificate against CA certifiates in TrustedCert-Dir.
- **TrustedCertDir:** A directory with certificate authority certificates to verify the server certificate.

3.1 Configuration of the Java command line client

The Java client also reads its configuration from a file, called by default `mdjavaclient.config`. This file is searched in the same places as the C++ API looks for the `mdclient.config`, that is, first in the current directory, then in `$HOME/.mdclient.config` and finally on `/etc/mdjavaclient.config`. Like the C++ client, only the port and server of the configuration file can be overridden on the command line of the clients:

```
mdjavaclient.sh [-p port] [hostname]
```

Next are the properties recognized on the `mdjavaclient.config` file. The following have the same syntax as in the C++ configuration.

- Host
- Port
- Login
- PermissionMask
- GroupMask
- Name:

The following exist only on the the Java configuration file or else have a slightly different syntax:

- **AuthMode**: Possible values are GridProxy, Certificate, Password, None. Default is None.
- **Password**: The user password. Only used when the AuthMode is set to Password. If not provided here, it must be provided at runtime by the user.
- **UseSSL**: Possible values are 1 for enable, 0 for disable. Default is 0. Needed for any authentication using certificates and grid proxies. If you use SSL the entire session will be encrypted. Some servers may require you to use SSL to connect.
- **CertFile**: Path to your x509 certificate in .pem format. For this option to have any effect, you need also options UseSSL and the AuthMode set to Certificate.
- **KeyFile**: Path to your x509 private key in .pem format. If the key is encrypted you will be asked for the passphrase on client startup. For this option to have any effect, you need also options UseSSL and the AuthMode set to Certificate.
- **PrivateKeyPassword**: Password for private key. If not defined, the password must be provided at runtime.
- **VerifyServerCert**: Verifies the server certificate against CA certifiates in `TrustedCertDir`.
- **TrustedCertDir**: A directory with certificate authority certificates to verify the server certificate.

4 Metadata Access from the Shell

You can access the metadata catalogue either with the `mdclient` metadata terminal tool as configured in the last section, or via the `mdcli` (or `mdjavacli` for those using the java package) command line tool which allows you to directly issue metadata commands on the shell, it's output is intended to be easily parseable by scripting languages:

```
> mdcli -p8822 -slocalhost listattr /
t
text
f
float
```

Metadata(p. ??) commands are parsed into pieces which are each separated by white space similarly to shell commands. If you want the white space to be part of one piece of the command itself, for example when you want to set an attribute to a string which contains white space, you must enclose it in single quotes: ' '. Single quotes are part of the client-server protocol and used when parsing the commands into parts. You need them every time a part shall contain spaces. Double quotes however are used in queries **Metadata Queries**(p. 11) (expressions evaluated by the database backend) to distinguish strings from variable references and common values.

Note that quotes may be removed by the shell when parsing a shell command, so if you are using the `mdcli` tool where the AMGA command is given on the command line, you will need to protect these single quotes from being removed by the shell with double quotes: ". The various APIs will in contrast usually (that is the Python and Java APIs do so, but not the C-API) automatically quote any arguments you pass to them with single quotes so they are not to be used in those APIs. The following is an example with `mdclient`, `mdcli` and the Python API showing how quotes are being used:

```
> mdclient
Query> find /files '/files:producer="CERN"'
> mdcli find /files "'/files:producer="CERN"'"
An in python:
mdclient.find('/files', '/files:producer="CERN"');
```

The metadata server uses a streaming protocol. Some APIs (for example the Java one) allow to interrupt the streaming of a response. The same is true for the `mdclient`. Pressing **CTRL-C once during the transmission of the result** will interrupt the streaming of the result. Only pressing CTRL-C a second time will terminate the client.

In the following is given a list of metadata commands. Additionally commands may be available for group or user access management, as described in **Users, Groups and ACLs**(p. 17) or **Management of Users using the database backend**(p. 17) depending on the server set up. To find out which commands are available on the server you are connected to, use the `help` command.

4.1 Commands for entry manipulation

- **addentry entry (attr value)+**: Creates a new entry and assigns the given values to the provided attributes. Examples:

```
addentry /testdir/a id 10
addentry b id 10 finished 'Oct-10-2004'
```

Possible errors are:

- 3 Illegal command: Syntax error
 - 4 Permission denied: You need write permission on the directory to create a file in it.
 - 7 Illegal Key
 - 10 No such key
 - 15 Entry exists
- **addentries (entry)+**: Creates the given entries in the catalogue. This command is done in a transaction, that is either all entries are inserted or none. Entries can be spread over several directories. Example:

```
addentries a
addentries /test1/a /test2/a /test1/b
```

Possible errors are:

- 1 No such file or directory: You tried to insert into a directory which did not exist.
- 3 Illegal command: Syntax error
- 4 Permission denied

- **rm pattern**: Removes all files matching pattern, where pattern may only contain wild cards in the file part. In order to remove an entry you need write permissions on the parent directory.
- **listentries [directory/schema]**: Returns the name of all entries in a given directory or schema. This differs from the **dir** command in that it will not show any directories but also that it shows only entries attached to a schema in the case of an AMGA catalogue collaborating with a file-catalogue. The result is returned in the following way:

```

0
entry 1
...
entry n
EOT

```

- **upload dir (attribute)+**: Starts an upload of entries into the catalogue. Currently a static restriction of the prototype is that there can be only up to 98 attributes assigned like this. After the upload is initialized, the put, abort and commit commands are allowed. Errors are returned by the call immediately, the OK code is delayed till the entire upload is successfully committed.
- **put file (values)+**: Inserts a new entry during upload. Errors are returned by the call immediately, OK is delayed until upload is committed.
- **abort**: Aborts upload. Errors and OK are returned by the call immediately.
- **commit**: Commits upload. Errors and OK are returned by the call immediately.

4.2 Commands for Manipulating Attributes

- **addattr dir/entry (key type)+**: Adds new keys to the list of keys of a directory. In a relational database backend these keys become the columns of a table associated to a directory. You should only use one key/type pair currently for compatibility reasons because some older backends like PostgreSQL <=7.4 do not allow to alter a table adding several columns. Possible types are explained in **AMGA data types**(p.13). The type is only used as a hint for the back end to store the data efficiently and allow efficient queries. The type may be ignored by the implementation (e.g. if the back end is a filesystem). In a filesystem the types and defined keys could be stored as attributes of directories. Some storage backend may allow you to define keys on a per-entry basis. Possible errors are:
 - 1 No such file or directory
 - 4 Permission denied. You need write permission on the directory.
 - 7 Illegal Key
 - 9 Internal error: All kinds of errors like duplicate keys
- **removeattr dir/entry key**: Removes the attribute or key from the list of attributes of the directory dir or the directory of the given entry, or if the implementation allows it from the list of attributes of a given entry. Attributes can only be removed if they are not used by any entry. So you either have to remove all entries for which this key is set or use **clearattr** to set the value of the attributes to NULL In order to remove an attribute, write permissions on the directory are necessary. Possible Errors are:
 - 1 No such file or directory
 - 4 Permission denied. You need write permission on the directory.
 - 10 No such key
 - 14 Attribute in use

- `schema_create dir (attr type)+ [option]`: Creates a new directory with a given schema. This is an atomic replacement for a sequence of `createdir` and `addattr`. The meaning of the optional `option` argument is backend dependent and you should not use it if you want to retain this independence. With a MySQL backend you can give here the name of the table engine, for PostgreSQL the keyword `inherit` will make the table inherit its schema from the parent directory.
- `setattr file (attribute value)+`: Sets a list of attributes of a file to given values. The attributes must exist.
- `getattr pattern (attribute)+`: Returns the filename and all attributes in turn for every file matching `pattern`. The following output is returned on success:

```

0
file 1
attr 1
...
attr n
file 2
attr 1
...
attr n
...
file n
attr 1
...
attr n
EOT

```

- `listattr file`: Returns a list of all attributes and their types in the following format:

```

0
attr 1
type 1
attr 2
.
.
.
type n
EOT

```

- `clearattr path attribute`: Sets the attributes of all files matching `path` to NULL. `Path` may currently contain wildcards only in the file-part. On success returns 0.

4.3 Finding and Updating Entries

- `find path query`: Returns a list of filenames matching `path` and fulfilling the query with their attributes. The `path` may currently contain only wild cards in the file name part. Query must be enclosed in single quotes. Strings in the query must be quoted with double quotes. For the supported syntax see the chapter about **Metadata Queries**(p. 11). **WARNING:** Be careful with patterns which also match a subdirectory, the result is undefined. The result is returned in the following way:

```

0
file 1
.
.
.
file n
EOT

```

The following errors may occur: 8 Illegal query: The query could not be parsed or violates security rules. No further information is currently provided on the reasons for this error. 2 Connection to DB failed or syntax error in path or query

- **updateattr pattern (attribute value)+ condition**: Updates attributes of entries matching a pattern in a single collection based on a condition. The values to which the attributes are updated can contain attributes as variables. Complex expressions are allowed as values. The condition may reference attributes of other collections. Updates are atomic. Examples:

```
updateattr /testdir1/* events events+1 'events>100'
updateattr /testdir1/* events events+1 '/testdir2:key > 0'
```

The first example increases the number of events of every file in /testdir1 which has more than 100 events by one. The second example increases the number of events of every file in /testdir1 provided there is an entry in the collection /testdir2 which has the attribute "key" set to anything larger than 1 (usefull to do locking by clients: putting such an entry into /testdir2 would lock /testdir1). For the supported syntax for the queries see the chapter about **Metadata Queries**(p. 11).

- **selectattr (attribute)+ condition**: Selects attributes from several collections based on a condition doing an inner join on the collections based on a join condition. The FILE attribute is used to select the entry name of an entry.

```
selectattr /jobdir:FILE /configdir:id /jobdir:eventGen /configdir:id
'/jobdir:events>1000 and /configdir:key=/jobdir:key'
```

This selects the entry-name of a job, the id in the configuration, the event generator name of a job and the id in a configuration for all jobs and configurations where the job has more than 1000 events and the keys attributes of the jobs and configurations match. For the supported syntax for the queries see the chapter about **Metadata Queries**(p. 11).

4.4 Manipulating Collections

- **createdir /parentdir/dir [option]**: Creates the directory dir if it does not yet exist but parentdir already exists. The directory is created with the current owner and the same ACLs as the parent directory. The option filed is a comma seperated list of options (no spaces allowed). The following options are available as of AMGA 1.1:
 - **shared**: Subdirectories created under this directory share the same schema and database table of the parent directory
 - **acls**: Creates a directory with acls for every entry, this is currently only supported by PostgreSQL and only if the necessary supporting stored procedures have been installed first.
 - **InnoDB**: Only MySQL. Specify that you want an InnoDB table as the backend table, this allows for example GIS functionality.
- **dir [directory]**: Returns the name of all subdirectories and files in the directory. The result is returned in the following way:

```
0
entry 1
entry-type-1
...
entry n
entry-type-n
EOT
```

where the entry-type is either 'entry' or 'collection'. If AMGA collaborates with a file catalogue this command will effectively show the content of the file catalogue. If you want to see which entries have already been attached to a schema in the AMGA part use the listentries command.

- **stat [dir/entry]**: Returns information on a given entry or directory. They can also be a pattern in the case of several entries. You need read permission to get this information. For an entry the following is returned:

```
0
name
owner
owner-permissions
group-permissions
links
EOT
```

For a directory the following is returned:

```
0
owner
owner-permissions
EOT
```

- **rmdir path:** Removes all directories matching path. Directories are only deleted if they are empty and they have no attributes defined.
- **pwd:** Prints the current directory which you can change with cd.
- **cd path:** Changes the directory to the given path. Possible errors are:
 - 1 No such file or directory
 - 4 Permission denied

4.5 Permission Handling

- **whoami** Prints out the name of the current user. Note that this command does not need any connections of the AMGA server and can thus be also used to do a test on whether an AMGA server is alive and what response time it has.
- **chown entry/dir new_owner:** Changes the owner of a directory or entry. Only the owner of an entry is allowed to execute this, or the root-user. chown does not check whether the user exists, since user management is considered to be handled outside of AMGA (ideally). Possible errors are:
 - 1 No such file or directory
 - 4 Permission denied
- **chmod entry/dir new_permissions:** Changes the access permissions of an entry or directory. Entries have owner and group-permissions, while directories have owner permissions and group permissions are handled via ACLs. Group permissions for entries allow you to remove privileges granted for all entries in a directory via the directories ACLs. The format of new_permissions is **rwrxwx** for entries and **rw**x for directories where "-"-signs can be substituted for the letters if you do not want to give a certain privilege. The permissions for entries are the concatenation of first the user and then the group rights. The x-Flag allows a user to enter a directory or respectively list an entry. r-and w-flags allow users to read/write metadata while the w-flag for directories allows users to create or delete entries in the given directory. Users cannot list directories for which they don't have read permissions. The command works also for patterns and uses a transaction. Possible errors are:
 - 1 No such file/directory
 - 4 Permission denied

4.6 Index Management

- **createindex name collection '(attribute)+'** [algorithm]: Creates an index name on a collection directory using several attributes and a given algorithm. Algorithms depend on the backend. The index is later referred to /collection/name in **index_remove**.
- **index_remove index_remove /path:** Removes an index.

4.7 Backing Up Data

- `dump [dir]`: Recursively dumps the contents of a directory and all subdirectories so that they can be recreated by calling the sequence of AMGA commands printed out.

Only root is allowed to use this command. Possible errors are:

- 4 Permission denied

- `user_dump [dir]`: Dumps the contents of a the user database such that it can be recreated from the sequence of AMGA commands printed out.

Only root is allowed to use this command. Possible errors are:

- 4 Permission denied

- `grp_dump [dir]`: Dumps out the information on all existing groups so that they can be recreated by calling the sequence of AMGA commands printed out.

Only root is allowed to use this command. Possible errors are:

- 4 Permission denied

AMGA also has special commands for user and group management. They are optional and may not be available on your installation for example if it collaborates with a file catalogue and uses the permission system of that catalogue. For more information see **Users, Groups and ACLs**(p. 17) and **User Management**(p. 15).

5 Metadata Queries

AMGA provides its own query language which is similar to the SQL query language. It tries to offer a large subset of the common functionality of database systems in a transparent way to the user. The biggest difference to SQL is that in AMGA's query language tables are referred to as references to directories. AMGA will ensure access restrictions on the data so that users cannot infer data from queries if they have no read-access to that data.

Queries are performed in the following AMGA commands:

```
find entry_pattern query_condition
selectattr column_1_query ... column_n_query query_condition
updateattr attr_1 update_query_1 .... attr_n update_query_n query_condition
```

A query condition is a query which returns a boolean in order to select or not select an entry for retrieval or update. Examples are

```
/jobdir:events>1000 and /configdir:key=/jobdir:key
like(/jobdir:FILE, "t%")
```

Query conditions are used in the WHERE statements of the SQL queries which are passed to the backends.

The other queries used in the AMGA commands return general values which are returned to the user in the `selectattr` command or which are used to update attributes in the `updateattr` command.

Queries can contain either literal values like numbers or strings which are marked by double quotes. Make sure to use single quotes around double quotes if the string contains spaces. Queries can also contain attributes which are evaluated in the query by filling in the values of the attributes for the current value. In AMGA all queries are in fact inner joins over all tables mentioned in any of the queries of command, that is the all possible combinations of all entries of all tables are made and those selected matching the query condition. Inner joins are the most common type of join. Some database systems provide no other kind.

References to attributes take the form:

```
<directory>:<attribute>
```

where relative paths to the directory (which is synonyme for table or schema, here) are allowed. Examples are:

```
selectattr /test:t 'like(t, "Test%")'
selectattr count(/test:t) 'like(t, "Test%")'
```

From the above example you can see that also functions are allowed. Function names are case-sensitive and lowercase. The following functions are available:

- `lower(string)`: Converts string to lower case.
- `upper(string)`: Converts string to upper case.
- `count(x)`: Aggregate function. Counts how often the attribute is set (not = NULL)
- `abs(x)`: Absolute value of x.
- `sin(x)`: The sine of x.
- `cos(x)`: The cosine of x.
- `tan(x)`: The tangens of x.
- `atan(x)`: The arc-tangens of x.
- `sqrt(x)`: The square root of x.
- `log(x)`: The natural logarithm of x.
- `rnd()`: A random number between 0 and 1.
- `sum(x)`: The aggregate sum of x.
- `max(x)`: The aggregate maximum of x.
- `min(x)`: The aggregate minimum of x.
- `avg(x)`: The aggregate average of x.
- `length(string)`: The length of the string.
- `pow(x, y)`: x to the power of y.
- `mod(x, y)`: x modulo y.
- `concat(str1, str2)`: The concatenation of str1 and str2.
- `like(str, pattern)`: Whether str is like pattern. The pattern is an SQL90 pattern.
- `substr(str, n, m)`: The substring of length m of str starting at n.

Queries can contain the following operators: `+`, `-`, `*`, `/`, `=`, `and`, `or`, `not`, `>=`, `<=`, `<>` or `!=`

Special attribute names refer to the properties of an entry:

- `FILE`: The name of the entry.
- `LINK`: The link pointed to.
- `OWNER`: The owner of the entry.
- `PERMISSIONS`: The owner's permission.
- `GROUP_RIGHTS`: The group-rights. These names could e.g. be used to restrict access to entries using a `VIEW`.

The exact syntax of AMGA queries is described in the annotated `parser.y++` and `lexer.l++` sourcecodes.

6 AMGA data types

The AMGA metadata server supports a set of generic datatypes for all the possible backends. It is guaranteed that if you add an attribute of with one of these types that it is translated into a type supported by the database. AMGA also guarantees that a `listattr` command will return the same data type.

Other datatypes supported by a given back end can be used, however they will be not portable and it is not guaranteed that `listattr` will return the type you specified with `addattr`.

6.1 AMGA data types

The following table lists the generic metadata data types supported by AMGA and their internal representation in the back ends:

	PostgreSQL	MySQL	Oracle	SQLite	Python
int	integer	int	number(38)	int	int
float	double precision	double precision	float	float	float
varchar(n)	character varying(n)	character varying(n)	varchar2(n)	varchar(n)	string
timestamp	timestamp w/o TZ	datetime	timestamp(6)	unsupported	time (unsupp.)
text	text	text	long	text	string
numeric(p,s)	numeric(p,s)	numeric(p,s)	numeric(p,s)	numeric(p,s)	float

The datatypes have the following properties:

- **int**: at least 32 bit integer value.
- **float**: 64 bits IEEE double precision floating point number.
- **varchar(n)**: A string of up to at least $n = 254$ characters. The low limit of 254 is imposed by MySQL 4.0 and smaller version. MySQL may also truncate any trailing white space, see MySQL documentation.
- **timestamp**: Timestamp with format 'YYYY-MM-DD HH:MI:SS'.
- **text**: Long text string (2GB size limit).
- **numeric(p,s)**:SQL numeric type with precision p and scale s .

Note that SQLite not really has a strong typing system. SQLite is in fact typeless, so you can try to store anything into a given column, although internally it distinguishes between text strings and 64 bit double precision values. A column is of type text if the column type contains any of the following substrings: BLOB, CHAR, CLOB, TEXT.

The pure Python back end distinguishes strings, floats (64 bit), integers (64 bit) and timestamps.

Both pure python and SQLite will accept any type and guarantee you that the same type is returned by `listattr` with the default datatypes being numeric and string respectively.

7 Configuring the AMGA Server

The AMGA metadata server `mdservermt` is configured using the `mdservermt.config` file.

7.1 Format of the Configuration File

Every line of the file contains a key value pair separated by an equal (=) sign. Blank space is ignored and keys and values may not contain white space. Lines can be continued with a backslash (\). Comments can be put at the beginning or end of a line after a hash-sign (#). The escape character is the caret (^). The following escape sequences exist:

- `^#` : Gives #
- `^^` : Gives ^
- `^\ : Gives \`
- `^ (^space)`: Insert a space.

7.2 Example of a server configuration file

```
# Server options
Port=8822 # Default 8822
MinProcesses = 2
MaxProcesses = 50 # Default: 50
# MaxConnectsPerProcess = 1000000

# Database options
# The DataSource option must match a data source name in the odbc.ini file
#DataSource=sqlitedb
#DataSource=Oracle10g
DataSource=PSQL
DBUser=arda
#DBPass=xxxxxxx

# Session options
Sessions = allow # Values are: no allow force
IdleTimeout = 1200 # Timeout for a connection/query in sec [20m default]
SessionTimeout = 86400 # Timeout for a session in the cache in sec

# Secure Connections
UseSSL = 1

# Authentication options
RequireAuthentication = 0 # If this is off, no authentication is done!
AllowCertificateAuthentication = 1
AllowPasswordAuthentication = 1
# If you use SSL, you need to load server certificates:
CertFile = cert.pem
KeyFile = key.pem
# If Certificate based authentication allowed, you need to load server certs
# TrustedCertDir = /etc/grid-security/certificates
# AllowGridProxyLogin = 1 # Requires also AllowCertificateAuthentication

# Authorization options, choose 0 or more
#MapFile = /etc/grid-map-file # Authorization based on certs
# Authorization based on certificates, put a list of VOMS URL and assigned users, here:
#VOMSGroups = https://lcg-voms.cern.ch:8443/voms/lhcb/services/VOMSAdmin?method=listMembers, lhcb, \
# https://kuiken.nikhef.nl:8443/voms/picard/services/VOMSAdmin?method=listMembers, picard
UserDB = 1 # Authorization based on certs & passwords
#VOGroupMap =
#VirtualOrganizations = gildav(gilda) # vo1(defaultUser1), vo2, vo3(defaultUser2)...
#OnTheFlyVOGroups = 1 # Whether groups are created every time on the fly
# if someone with a VO-cert logs in
#VOGroupMap = gildav:/gildav(gildav:users)
#VOUserMap = gildav:/gildav/Role~=TrailersManager(gildav)
#MyProxyHack = 1 # Allow roles in MyProxy certificates for login
```

7.3 Description of the server options

The following options are supported:

- **Port:** The number of the port the server will listen on. This can be overridden on the command line with the `-p` switch. The default port is 8822.
- **MinProcesses:** This is the minimum number of processes waiting for client connections the server must offer. When the server starts up or there are no client connections for some time, `MinProcesses` is the number of processes spawned waiting for connections.
- **MaxProcesses:** This is the maximum number of processes the server will spawn in total. The server always tries to have 1/3 of the processes in the awaiting connection state. To achieve this, the server will spawn new processes until the number of `MaxProcess` is reached. Please make sure that your database backend can support as many client connections.
- **MaxConnectsPerProcess:** To prevent any very rare memory leaks or other resource leaks to reduce the stability of the service, server processes can be asked to terminate themselves after serving a certain number of connections. The default is not to do this.

- **DataSource:** An ODBC database source which you need to have configured in an `odbc.ini` file. Note: you can use programs like `gODBCConfig` or `iodbcadm-gtk` to configure ODBC. The underlying database needs to be prepared for AMGA by running the `createInitial.sql` script in the `scripts` directory of the AMGA distribution.
- **DBUser:** The user with which the server will contact the database backend. It is possible to give this user name also in the `odbc.ini` file.
- **DBPass:** The password the server will give when contacting to the database backend. You can configure this also in an `odbc.ini` file.
- **Sessions:** This defines whether you want to allow sessions. Sessions create an overhead on the protocol if they are enforced, so the performance of individual clients may reduce while you will be able to support more clients which share the available connections (there is a maximum of `MaxProcesses` connections, if they are all hogged by a client, then no new clients will be able to connect). Such a denial-of-service situation can be prevented by forcing sessions. Values are: `no`, `allow`, `force`. Default is `allow`.
- **IdleTimeout:** Timeout for an idle connection (that is a connection that waits for a client command) in seconds. There are no timeouts currently for database queries apart from how the database is configured. The default is 20 minutes. This is what will make your `mdclient` command line tool time out.
- **SessionTimeout:** Timeouts for session. The lifetime of a session in seconds. Default is 1 day.
- **UseSSL:** Whether the server will offer SSL as a connection protocol. This is also required to allow certificate based authentication and if you want to use passwords this is recommended if you want to be sure no one listens in. Values are 0 and 1, default is 1. Note that you cannot force the client to use an SSL connection.
- **RequireAuthentication:** Whether users need to be authenticated. Default is 0: `no`.
- **AllowCertificateAuthentication:** Whether you allow users to authenticate with their certificate. You will need to have CA certificates loaded for the server for this to work in order to be able to verify the client's certificate. See **TrustedCertDir**. You should also look at **User Management**(p.15) because you will need to have a user manager module running for this to work. Default is 0: `no`.
- **AllowPasswordAuthentication:** Allow authentication with a password. You need a user manager module running for this to work. See **User Management**(p.15) . Default is 0: `no`.
- **CertFile:** The path to the server certificate in PEM format. Preferably without encryption. Necessary to use SSL.
- **KeyFile:** The path to the private key of the server in PEM file format. Necessary to use SSL.
- **TrustedCertDir:** Path to a directory with trusted CA certificates. Needed to verify a client certificate.
- **AllowGridProxyLogin:** Whether you allow users to authenticate with a proxy certificate. Default is 0: `no`.

For the rest of the options see **User Management**(p.15) .

8 User Management

The standalone AMGA server comes with a powerful system to manage users as well as to control access to entries and metadata. If AMGA is run as an add on to a file catalogue, however, these features are not available and the access controls of the file catalogue is used instead.

To understand the user management of the AMGA server it is necessary to know that the server does not really manage users but only their authentication and authorization. When changing the owner of an entry for example, the server does not check that this owner exists. Users are only relevant for logging in. This allows to manage users outside of the server, e.g. in a VOMS.

8.1 Configuration

To use the metadata service, a user must be authenticated and authorized. Authentication can be done via a certificate or a password, see **Configuring the AMGA Server**(p.13) . After the authenticity of a user is established in the handshaking of the client with the server, the client needs to be authorized to use the role of a certain user. Authorization is optional, if authorization is not enabled for the server, any authenticated user can assume any role he wishes. Authorization is controlled via the `mdserver.config` configuration file:

```
# Authorization options, choose 0 or more
# MapFile = /etc/grid-map-file          # Authorization based on certs
# Authorization based on certificates, put a list of VOMS URL and assigned users, here:
VOMSGroups = https://lcg-voms.cern.ch:8443/voms/lhcb/services/VOMSAdmin?method~=listMembers, lhcb, \
             https://kuiken.nikhef.nl:8443/voms/picard/services/VOMSAdmin?method~=listMembers, picard
UserDB = 1 # Authorization based on certs & passwords
```

Authorization can be done via certificates or passwords (password authentication actually includes authorization), both must be explicitly enabled. **For authentication via certificates to work, both the server and the client must have SSL enabled (UseSSL)**. Four ways are foreseen to accessing the necessary information to match user names with their credentials, one or more must be enabled for the `RequireUserAuthorization` to work:

- A grid-map file mapping certificate subjects, that is distinguished names (DN of users) to users. This is a static setup and no new users can be added at runtime. No password authorization is possible via a grid map file. Option `MapFile`.
- A user database using the database backend. This allows creation of users and the management of their credential at runtime. This is the only option which allows password based login. Option `UserDB`.
- Authorization using a VOMS. All users registered with a VO will be assigned to the user specified here. You can give several VOMS-URL user pairs here. Option `VOMSGroups`.
- Authorization via VOMS certificates. All users connecting with a VO information-enriched certificate obtained via `voms-proxy-init` will be assigned to specific AMGA users depending on the role within the VO. Option `VirtualOrganizations`. Note that only the DB based user management module is able to make changes to the user setup. If you have several user management modules activated at the same time, then listing users and checking their credentials for authorization will go through the users in all of the modules. A user is authorized as soon as he has been found in **any** of the modules.

8.2 Management via a Grid Map file

You can give a location of a grid map file using the `MapFile` option for user authorization. This file contains pairs of distinguished names and user names. The DN must be enclosed in double quotes and must be in the form where its fields are separated by commas on one line (output of `openssl x509 -subject -in usercert.pem -nameopt oneline -noout`):

```
> cat mapfile
"/C=CH/O=CERN/OU=GRID/CN=Birger Koblitz 9904" koblitz
```

There are no wild cards currently allowed. The map file will be read only once at server startup. It is not possible to add or change users using the command line tool.

8.3 Management of Users using the database backend

To enable user management using your database backend, you need to enable this feature by setting `UserDB = 1`. If you have run the `createInitial.sql` script, to set up your database, the necessary tables have already been created. You can now manage users via the `mdclient` command line tool:

- `user_list`: Lists all users known to the authentication subsystem.
- `user_listcred user`: Lists the credentials with which the user can be authenticated. Returns first the user name, then whether there is a password and then the certificates which are mapped in a Grid-Mapfile and via the user database. Finally the different VO and VO roles which allow you to become that user are listed. Only root is allowed to see the credentials of other users.
- `user_create user [password]`: Creates a new user and assigns a password if given. This command is for the root user only. Only hashes of passwords are stored in the database backend.
- `user_remove user`: Deletes a user. This command is for root only. It does not check whether there are still files or directories owned by that user.
- `user_password_change user password`: Changes the password of a user. Only root can change the password of any user. Non-privileged users may only change their own passwords.
- `user_subject_add user subject`: Adds a certificate identified by its subject line to be used to authenticate a user. While every user can only have one password. Several certificates can point to the same user. Remember that in order to have spaces in the subject, you need to enclose it by single quotes ("). See **Definition of the Client Server Protocol**(p. 23). The form of the subject needs to be the one where parts are separated by commas as in the output of e.g. `openssl x509 -subject -in usercert.pem -nameopt oneline -noout`.
- `user_subject_remove user subject`: Removes a certificate from the list of certificates which allow to login as a certain user.

8.4 Management via a VOMS

Giving pairs of VOMS member list URLs and user names in the `VOMSGroups` option, you can assign all members of a VO to a user (role would be the better word, here).

8.5 Management via VO-Certificates

You can give allow users to log in with VO-enabled certificates by using the `VirtualOrganizations` option and assigning it a list of `VO(default_user)` definitions. By enabling `MyProxyHack` this works also with certificates issued by a MyProxy server. The `VOGroupMap` and `VOUserMap` options allow to map VO groups to AMGA groups and special VO roles to AMGA users with the syntax used by `VirtualOrganizations`.

9 Users, Groups and ACLs

The standalone AMGA server comes with a powerful system to manage users as well as to control access to entries and metadata. If AMGA is run as an add on to a file catalogue, however, these features are not available and the access controls of the file catalogue is used instead.

9.1 Users

The size of a username is limited to 64 lower-case latin alphabet characters.

9.2 Groups

Any user can create groups. Group names are scoped with the name of the user creating them. A fully qualified group name has the form `user:groupname`. If the user scope of the group is the current user, it does not need to be specified in a command. The size of `groupname` is limited to 64 lower-case latin alphabet characters.

A special group exists and is maintained by AMGA internally, the `system:anyuser` group which contains automatically any user which is authenticated to the system. Using this group it is possible to emulate the permissions for 'other'-users in a Unix filesystem which are missing in AMGA.

The following commands can be used to manage groups:

- `grp_create groupname`: Creates a new group with name `groupname`. It is not possible to create groups belonging to others.
- `grp_delete groupname`: Deletes a group with name `groupname`. Only root can delete groups of other users.
- `grp_show groupname`: Shows all the members belonging to group `groupname`. You can only look into groups of which you are a member or your own groups. Root can list all groups.
- `grp_adduser groupname user`: Adds a user to a group. Only owners of a group or root can change group membership.
- `grp_removeuser groupname user`: Removes a user from a group. Only owners of a group or root can change group membership.
- `grp_member [user]`: Shows to which groups a user belongs. Only root can ask this question for other users.
- `grp_list [user]`: Shows the groups owned by `user`, by default the current user. Only root can list other user's groups.

9.3 Access Control Lists

ACLs (Access Control Lists) can be assigned to any directory.

The following commands exist to manipulate ACLs of a directory.

- `acl_add directory group rights`:
- `acl_remove directory group`:
- `acl_show directory`:

10 Installation from Source

To install the ARDA metadata server from source you will need to first download the source distribution from the `download` directory.

For compilation you need to install a development package for ODBC (e.g. `unixodbc`) This should be part of any standard distribution. On CERN SLC3 you simply should be able to do:

```
apt-get install unixODBC unixODBC-devel
```

You will then need the `libxml2` development library, which should also be part of any distribution. On SLC3 you can simply install it using

```
apt-get install libxml2-devel
```

For the server you will finally need to install the boost libraries:

```
apt-get install boost-devel
```

The SOAP based service will in addition need gSOAP . You can directly download the binary package for Linux. It suffices to unpack to /opt to immediately get started.

Now you should be ready to compile and install the AMGA server:

```
tar xvfz glite-amga-server-1.1.0.tar.gz
cd glite-amga-server-1.1.0
./configure
make
su
make install
```

You will need to get at least one of the currently supported 4 database backends installed, including their ODBC driver. You have the choice among PostgreSQL, MySQL, Oracle and SQLite:

- **PostgreSQL**. This is the easiest solution, since ODBC drivers are included in any distribution package. On SLC3 you can install the Postgres ODBC driver using

```
apt-get install postgresql-odbc
```

PostgreSQL needs a little setup: You need to create a database and a user. Access should be made possible via TCP/IP from localhost. The `scripts/init-arda-psqldb.sh` script should be able to do this for you.

- **MySQL**. Again, ODBC drivers are included in any distribution. On SLC3 you can install the MySQL ODBC driver using

```
apt-get install MyODBC
```

Again, make sure you have a user created which gets access to the database on MySQL.

- **SQLite** is a file-based database. You need to get the ODBC driver and compile if it is not part of your distribution.
- **Oracle** can be used from CERN. You need to get the instant client and ODBC driver from Oracle. Which you can freely download after a registration [here](#). . You will need to set up Oracle for the service names at you place. At CERN you might try to learn something from the [Oracle Linux pages](#).

Examples of ODBC configuration files can be found in the `scripts` directory. Copy the `odbc.ini` and `odbcinst.ini` configurations into `/etc` or into your home directory (but then called `.odbc.ini` and `.odbcinst.ini`). In `odbc.ini` you need to configure the database used by the server and which server is being connected to. Examples are given for all 4 databases. The users (and passwords if required) must then be setup in the `mdserver.config` file. The ODBC configuration can be checked with e.g. `gODBCConfig` or `DataManagerII` which are probably installed along the ODBC package or other ODBC clients like OpenOffice. If you don't know about ODBC, some more hints can be found in <http://www.unixodbc.org/doc/User\~Manual/> the Unix ODBC User Manual .

Some initial tables need to be setup in the database. You need to run one of

```
sqlplus user/passwd@endpoint <scripts/createInitial.sql
psql -User database <scripts/createInitial.sql
sqlite3 dbfile.db <scripts/createInitial.sql
mysql database <scripts/createInitialMySQL.sql
```

to setup the database.

You should now proceed to configure the server (**Configuring the AMGA Server**(p.13)) and start it up.

The following database and ODBC versions are known to work:

- SQLite 3.2.1, but not 3.2.7 with the 0.64 and 0.65 ODBC drivers. In 3.2.7 the ODBC driver is incompatible to the library.
- MySQL 4.0.x and 4.1.x work, 3.x does not.
- PostgreSQL works starting from version 7.2, however only one attribute can be added at a time in versions before 8.0. Explicitely tested were versions 7.2, 7.3, 7.4 and 8.0 allways in their latest sub-versions.

11 Using the C++ Client API

There are two different C++ client APIs available for the AMGA metadata service. One is through the `md_api` which provides several api functions, the other is directly through the `MDClient` class which also serves as a backend to the `md_api`.

The `MDClient` class offers an interface which allows to issue AMGA commands directly but does not understand the semantics of the commands and thus does not parse the responses of the server into suitable structures, while this is done by the `md_api`. However, the control on the connection to the server is much better in the case of the `MDClient` class, for example it allows you to abort a query easily. It may also happen that some commands are not available in the `md_api` yet.

In any case, both ways to access the metadata service from C++ depend on an existing and accessible `mdclient.config` file being either in the current working directory or in the home directory as `~/mdclient.config`. See **Configuration of the C++ and Java command line clients**(p. 3) for explanations how to set up the client configuration.

The following is an example of a program using the `md_api` to

```
#include "client/md_api.h"
#include <iostream>

int main (int argc, char *argv[])
{
    std::cout << "Listing attributes of /test\";
    std::list< std::string > attrList;
    std::list< std::string > types;
    if( (res=listAttr("/test", attrList, types)) == 0){
        std::cout << " Result:" << std::endl;
        std::list< std::string >::iterator I=attrList.begin();
        while(I != attrList.end())
            std::cout << " >" << (*I++) << "<" << std::endl;
    } else {
        std::cout << " Error: " << res << std::endl;
    }

    std::cout << "Getting gen and events attributes of /test/*\n";
    AttributeList attributeList(2);
    std::list< std::string > attributes;
    attributes.push_back("gen");
    attributes.push_back("events");
    if( (res=getAttr("/test/*", attributes, attributeList)) == 0){
        std::cout << " Result:" << std::endl;
        while(!attributeList.lastRow()){
            std::vector< std::string > attrs;
            std::string filename;
            attributeList.getRow(filename, attrs);
            std::cout << "File: >" << filename << "<" << std::endl;
            for(size_t i=0; i< attrs.size(); i++)
                std::cout << " >" << attrs[i] << "<" << std::endl;
            std::cout << std::endl;
        }
    } else {
        std::cout << " Error: " << res << std::endl;
    }

    return 0;
}
```

A full overview of the available API functions is given at http://project-arda-dev.web.cern.ch/project-arda-dev/metadata/md_-\-\-api_-\-8cc.html

```
#include <MDClient.h>
#include <iostream>

int main (int argc, char *argv[])
{
    int res;

    MDClient client;
    // client.setDebug(true);
```

```

if(client.connectToServer()){
    std::cout << client.getError() << std::endl;
    return 5;
}

std::string command="pwd";
if( ( res=client.execute(command) ){
    std::cout << " ERROR: execute failed"
        << " (" << res << "): "
        << client.getError() << std::endl;
    return res;
}

while(!client.eot() {
    std::string row;
    if(res=client.fetchRow(row)){
        std::cout << "Error fetching: " << res << std::endl;
        return res;
    }
    std::cout << row << std::endl;
}

return 0;
}

```

All capabilities of the `MDCClient` like cancellation of requests or the catching of `CTRL_C` are explained in the reference at <http://project-arda-dev.web.cern.ch/project-arda-dev/metadata/class\MDCClient.html> a short(!) example of how to make use of them is the `mdclient.cc` program itself.

12 Using the Java Client API

The AMGA Java API is distributed in two forms. As an RPM and as a tar ball. The tar ball is provided so that the Java API can be used in other platforms other than Linux, including Windows and MacOS.

To use the Java API it is necessary to include the `glite-amga-api-java.jar` file in the classpath. If the Java API was installed from the RPM, then this file is typically located at `<GLITE_HOME>/share/java`, where `<GLITE_HOME>` is the base directory where the `gLite` software is installed (typically, `/opt/glite`). If the Java API was installed directly from the tar ball available on the AMGA Web Site, the `glite-amga-api-java.jar` is located on the top level directory to where the tar ball was unpacked.

A jar can be included in the classpath in two ways: by setting the `CLASSPATH` environment variable or by using the `-classpath` option in the command line arguments when running `java`.

To set the classpath variable:

- Unix (bash): `export CLASSPATH=./glite-amga-api-java.jar`
- Windows: `set CLASSPATH=./glite-amga-api-java.jar`

After setting the `CLASSPATH`, to run a Java program it is only necessary to do the following to run a class called, for instance, `QueryMetadata`:

```
java QueryMetadata
```

To specify the jar file directly on the command, one must do:

Unix:

```
java -classpath ./glite-amga-api-java.jar QueryMetadata
```

On Windows the command line is similar, except the path separator is `;` instead of `.`.

The Javadocs for the Java API can be found here <http://project-arda-dev.web.cern.ch/project-arda-dev/metadata/java>.

Like the C++ API, the Java API can be used in two ways. Either through the higher-level interface exposed by the class `arda.md.javaclient.MDCClient` or by sending the commands

directly to the server using the low-level API in `arda.md.javaclient.MDServerConnection`. Next are the two examples given for the C++ client API rewritten using the Java API. The first uses the higher-level Java API.

```
import java.io.IOException;
import arda.md.javaclient.*;

public class MDJavaAPI {

    public static void main(String[] args) throws IOException {
        MDServerConnection serverConn = new MDServerConnection(
            MDServerConnectionContext.loadDefaultConfiguration());
        MDClient mdClient = new MDClient(serverConn);

        System.out.println("Listing attributes of /test");
        try {
            AttributeDef[] attrs = mdClient.listAttr("/test");
            System.out.println("Result: ");
            for (int i = 0; i < attrs.length; i++) {
                System.out.println(" >" + attrs[i].name + ":" + attrs[i].type);
            }
        } catch (CommandException e) {
            System.out.println("Error: " + e.getMessage());
        }

        System.out.println("Getting gen and events attributes of /test");
        try {
            String[] keys = {"gen", "events"};
            NamedAttributesIterator attrs = mdClient.getAttr("/test", keys);
            while (attrs.hasNext()) {
                NamedAttributes entry = attrs.next();
                System.out.println("File: " + entry.getEntryName());
                String[] keys1 = entry.getKeys();
                for (int i = 0; i < keys1.length; i++) {
                    System.out.println(" >" + keys1[i] + "=" + entry.getValue(keys1[i]));
                }
            }
        } catch (CommandException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

The following example uses the low-level API directly:

```
import java.io.IOException;
import arda.md.javaclient.*;

public class DirectServerConnection {

    public static void main(String[] args) throws IOException
    {
        // Loads default configuration and connects to server
        MDServerConnection serverConn = new MDServerConnection(
            MDServerConnectionContext.loadDefaultConfiguration());
        try {
            serverConn.execute("pwd");
            while (!serverConn.eot()) {
                String row = serverConn.fetchRow();
                System.out.println(">" + row);
            }
        } catch (CommandException e) {
            System.out.println("Error executing command: " + e.getMessage());
        }
    }
}
```

13 Using the Python Client API

The Python client for AMGA is distributed in the `glite.amga.api-python` RPM. After installation the `amga` package is available with the `mdclient` and `mdinterface` modules. The

mdclient class offers an interface similar to the MDClient interface in C++ plus methods for most AMGA command. All arguments are automatically quoted before being sent to the server.

The following is an example script which creates a directory, cd's into it and then gets the "sin" and "events" attributes of all entries in the directory.

```
#!/usr/bin/env python

#import amga classes
from amga import mdclient, mdinterface

#instantiate an AMGA client connecting to localhost:8822 as 'guest'
client = mdclient.MDClient('localhost', 8822, 'guest')

try:
    print "Creating directory /pytest ..."
    client.createDir("/pytest")
except mdinterface.CommandException, ex:
    print "Error:", ex

try:
    print "cd /pytest"
    client.cd("/pytest")
except mdinterface.CommandException, ex:
    print "Error:", ex

try:
    print "Getting all attributes of the files in /pytest..."
    client.getattr('/pytest', ['sin', 'events'])
    while not client.eot():
        file, values=client.getEntry()
        print "->",file, values
except mdinterface.CommandException, ex:
    print "Error:", ex
```

14 Definition of the Client Server Protocol

The protocol is a streamed ASCII protocol which is line oriented. Three bytes are special control characters:

is the line-ending byte which needs to be attached to any line, EOT (004) is the end of transmission sent by the server after any response because server responses can have many (also empty) lines and CAN (030) is the cancel byte which can be sent out-of-band by the client to abort the request or inline in the servers response if during response processing an error occurs.

This defines the full client server protocol including the handshaking with four example commands(first one OK, second cancelled by user, third has an execution error, fourth is cancelled by the server):

SERVER	CLIENT
Greeting\n	
Protocol <protocol number>\n	
<space sep. list of serv opts>\n	<requested option>\n
	<more opts>\n
	\n
OK\n	
----- SSL handshaking if required -----	
SSL_accept()	SSL_connect()

	<command>
0\n	
<line 1 of response>\n	
...	
<line n of response>\n	
EOT	
	<command>
<error-code> <literal err.>\n	
...	CAN (Out-of-Band!)
CAN	
<abort error code>\n	

```
EOT
                                     <command>
<err-code> <comment>\n
EOT
                                     <command>
0\n
<line 1 of response>\n
...
CAN (timeouts, back end-error...)
<err-code> <comment>\n
EOT
```