MDS 2.2: Creating a Hierarchical GIIS

This document describes by example how to create a hierarchical GIIS for use by MDS 2.2.

The following topics are covered in this document:

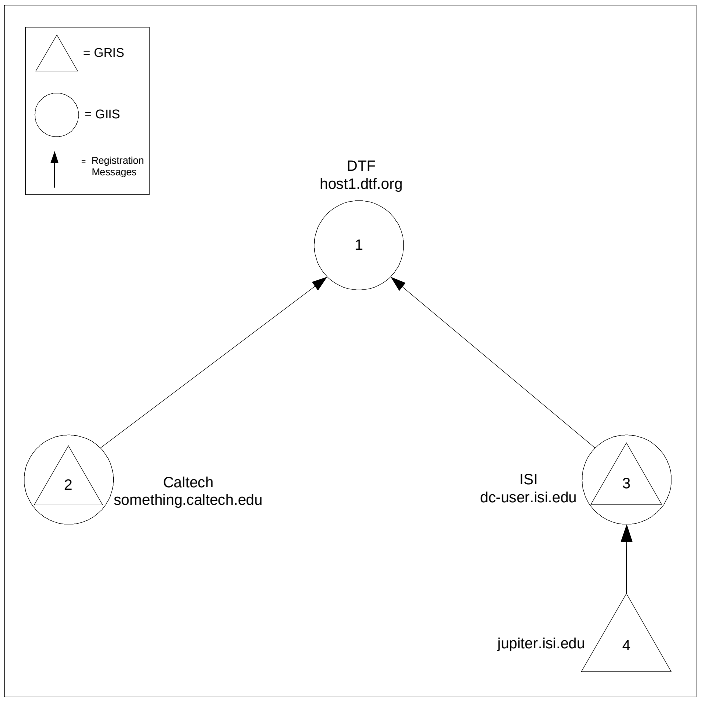
Configuration Files Used in Creating a Hierarchical GIIS Architecture

The essence of creating a hierarchical GIIS is to first decide on a hierarchical structure of GIIS servers, and then create the appropriate grid-info-slapd.conf files and grid-info-resource-register.conf files for each server in the hierarchy.

The grid-info-slapd.conf and grid-info-resource-register.conf files are processed by MDS when MDS is started with the following command: `$GLOBUS_LOCATION/sbin/globus-mds start`

This command starts the OpenLDAP 2.0 slapd server for the GRIS. The command does not require environment variables (GLOBUS_LOCATION).

The following example shows the details of creating a hierarchical GIIS. In this example we are creating three GIISs. At the top of the hierarchy is a GIIS named DTF. Then there are two "sub" GIISs. One is for Caltech, including a host called something.caltech.edu. The other is for ISI, consisting of two hosts – one called dc-user.isi.edu; the other called jupiter.isi.edu. Our sample GIIS hierarchy looks like this:



Now let's consider the configuration files for the hosts in the tree. Each host has a grid-info-slapd.conf file and a grid-info-resource-register.conf file.

As can be seen in the examples below, each grid-info-slapd.conf file should have only one suffix per database entry. The suffix describes the name of the object within the database. Each slapd instance can be considered to be a front end, connected with a number of back ends. Each back end must be represented by a single suffix.

In our example, a suffix would represent a host/server like DTF, ISI, or Caltech. A database would be something like giis, ldif, or ldbm as shown in the example files below.

Host 1: host1.dtf.org

grid-info-slapd.conf
```
  database: giis
  suffix:   mds-vo-name=dtf, o=grid
```

For the DTF GIIS, the grid-info-resource-register.conf file is not important because it is the root node in our sample hierarchy and does not need to send registration messages out to any other system.

Host 2: something.caltech.edu

grid-info-slapd.conf
```
  database: ldif
  suffix:   mds-vo-name=local, o=grid

  database: giis
  suffix:   mds-vo-name=caltech, o=grid
```

```
grid-info-resource-register.conf
  # Register Caltech vo with the DTF vo:
  dn:       mds-vo-op-name=register,mds-vo-name=dtf, o=grid
  reghn:    host1.dtf.org
  regport:  2135
  hn:       something.caltech.edu
  port:     2135
  rootdn:   mds-vo-name=caltech, o=grid

  # Register Caltech GRIS with Caltech GIIS:
  dn:       mds-vo-op-name=register,mds-vo-name=caltech, o=grid
  reghn:    something.caltech.edu
  regport:  2135
  hn:       something.caltech.edu
  port:     2135
  rootdn:   mds-vo-name=local, o=grid
```

Host 3: dc-user.isi.edu

```
grid-info-slapd.conf
  database: ldif
  suffix:   mds-vo-name=local, o=grid

  database: giis
  suffix:   mds-vo-name=isi, o=grid

grid-info-resource-register.conf
  # Register ISI vo with the DTF vo:
  dn:       mds-vo-op-name=register,mds-vo-name=dtf, o=grid
  reghn:    host1.dtf.org
  regport:  2135
  hn:       dc-user.isi.edu
  port:     2135
  rootdn:   mds-vo-name=isi, o=grid

  # Register ISI GRIS with ISI GIIS:
  dn:       mds-vo-op-name=register,mds-vo-name=isi, o=grid
  reghn:    dc-user.isi.edu
  regport:  2135
  hn:       dc-user.isi.edu
  port:     2135
  rootdn:   mds-vo-name=local, o=grid
```

Host 4: jupiter.isi.edu

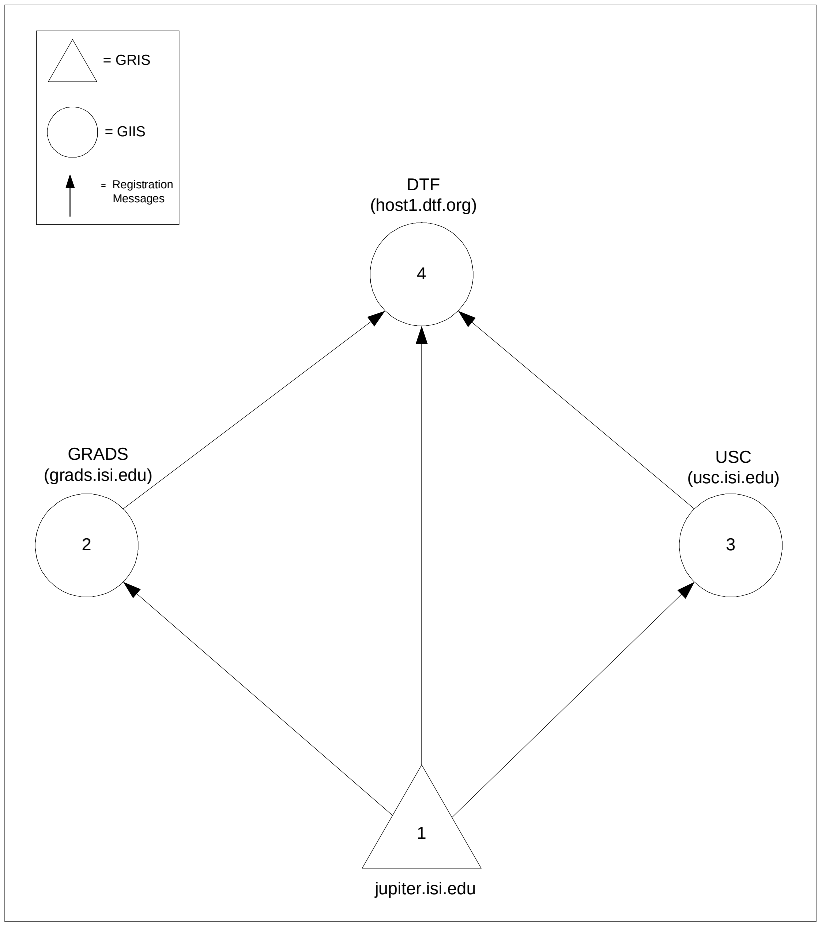
```
grid-info-slapd.conf
  database: ldif
  suffix:   mds-vo-name=local, o=grid

grid-info-resource-register.conf
  dn:       mds-vo-op-name=register,mds-vo-name=isi, o=grid
  reghn:    dc-user.isi.edu
  regport:  2135
  hn:       jupiter.isi.edu
  port:     2135
  rootdn:   mds-vo-name=local, o=grid
```

Renaming of Distinguished Names (DNs) in a GIIS Hierarchy

As the data from an information provider propagates up through a hierarchy of GIIS servers, the dn is renamed to enable appropriate query functionality.

Here is an example with a different topology to illustrate renaming functionality:

Suppose we have a hierarchical GIIS consisting (from bottom to top of the hierarchy) of a GRIS on jupiter.isi.edu, a GIIS called GRADS (grads.isi.edu), a GIIS called USC (usc.isi.edu), and a GIIS called DTF (host1.dtf.org). The following diagram illustrates this hierarchy:



Now let’s examine the data returned when querying for the MdsCpu data object on jupiter.isi.edu at the various levels of the hierarchy using the grid-info-search command.

1) query:
```
GLOBUS_LOCATION/bin/grid-info-search -x -h jupiter.isi.edu -p 2135 -b
"mds-vo-name=local,o=grid" -s sub "(objectclass=*)" "MdsCpu"
```

   result:
```
dn:  Mds-Host-hn=jupiter.isi.edu,Mds-Vo-name=local,o=grid
[ Object Data … ]
```

2) query:
```
GLOBUS_LOCATION/bin/grid-info-search -x -h grads.isi.edu -p 2135 -b
"mds-vo-name=GRADS,o=grid" -s sub "(objectclass=*)" "MdsCpu"
```

   result:
```
dn:  Mds-Host-hn=jupiter.isi.edu,Mds-Vo-name=GRADS,o=grid
[ Object Data … ]
```

3) query:
```
GLOBUS_LOCATION/bin/grid-info-search -x -h usc.isi.edu -p 2135 -b
"mds-vo-name=USC,o=grid" -s sub "(objectclass=*)" "MdsCpu"
```

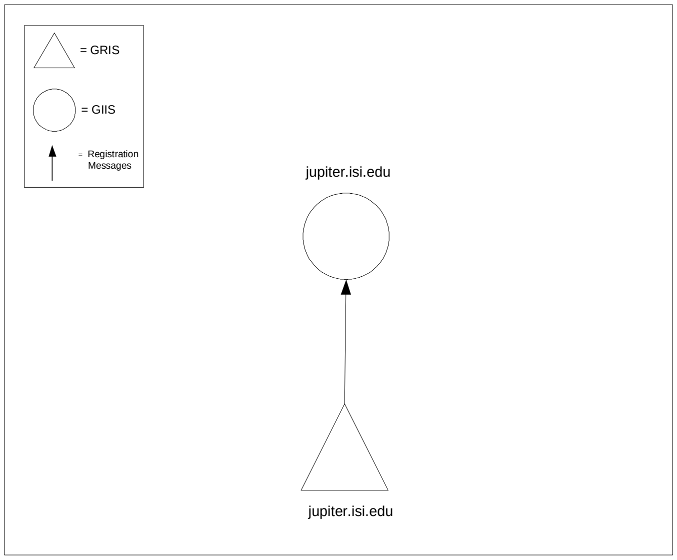
result:
```
dn:  Mds-Host-hn=jupiter.isi.edu,Mds-Vo-name=USC,o=grid
[ Object Data … ]
```

4) query:
```
GLOBUS_LOCATION/bin/grid-info-search -x -h host1.dtf.org -p 2135 -b
"mds-vo-name=DTF,o=grid" -s sub "(objectclass=*)" "MdsCpu"
```

result:
```
dn:  Mds-Host-hn=jupiter.isi.edu,Mds-Vo-name=DTF,o=grid
[ Object Data … ]

dn:  Mds-Host-hn=jupiter.isi.edu,Mds-Vo-name=USC,Mds-Vo-name=DTF,o=grid
[ Object Data … ]

dn:  Mds-Host-hn=jupiter.isi.edu,Mds-Vo-name=GRADS,Mds-Vo-name=DTF,o=grid
[ Object Data … ]
```

Timing Issues and Registration Control

Consider the default installation and setup of MDS, where one GRIS is registered to one GIIS. For the default configuration, the GRIS and the GIIS are on the same machine, so they have the same hostname as shown in the following example. Both the GRIS and GIIS are queried on port 2135; you differentiate between the two by setting mds-vo-name=local and mds-vo-name=site during a query.



As seen earlier in this document, the grid-info-resource-register.conf file can list one or more GIIS servers to which a GRIS will register directly. The parameters in this file identify host names, ports and several time values that control the registration messages from a GRIS to a GIIS server.

One grid-info-resource-ldif.conf parameter and several grid-info-resource-register.conf parameters (all in the GLOBUS_LOCATION/etc directory) are of value in ensuring correct query registration and complete return of data. These parameters are described in the paragraphs below. Refer to *Appendix A, Registration Configuration File Reference*, for a complete listing of registration control parameters and an example file.

Note that all configuration files used by MDS are described in *MDS 2.2 Configuration Files*.

LDAP sizelimit

In the discussion below, note that "client" refers to anything that queries MDS. This can be the ldapsearch command, the grid-info-search command, or the user's client tools. "Server" refers to the machine receiving the MDS query. A GIIS can be either a client or a server.

The LDAP sizelimit parameter resides in both the grid-info-resource-ldif.conf and grid-info-slapd.conf files. The sizelimit parameter defines the maximum result set size in number of objects returned by the server for any given client request. The client should specify a nonzero sizelimit appropriate to the specific server. If the client specifies a sizelimit of 0, the server will return all results, restricted by the server's maximum and default sizelimit values specified in grid-info-slapd.conf. If the client does not specify a sizelimit, the server may set the sizelimit to a default value.

If the client does not receive all expected results from a search sent to the server, one thing to check is that the sizelimit specified by the client (or that specified by the server default) is large enough to allow the return of all requested objects. The sizelimit should be set appropriately for the type of server and amount of data expected from the queries, so that timeouts do not occur.

Registration period (regperiod) and ttl

Since a GIIS can be either a client or a server, this section and the *Cache ttl* section following define "registrar" as the machine receiving registration data, and "registrant" as the machine sending registration data.

The regperiod and ttl parameters are both in the grid-info-resource-register.conf file. The registration period is the notification time for service availability. That is, regperiod specifies how often the registrant sends out a message saying that it exists. The ttl specifies how long the receiving registrar should keep the registration information. A general recommendation for the ttl value is twice the registration period. This will reduce the potential problem of building a GIIS but not getting all the expected results in cache.

For example, let's say we have a GRIS that sends out a message saying it exists, and it does this every minute. Then we have a GIIS that keeps this message for two minutes. If there is a network glitch and the GIIS misses one existence message, it still knows that the GRIS exists. This is because ttl = 2x regperiod in this case. If ttl=regperiod or ttl<regperiod, the existence message from the GRIS could be missed, and it would appear to the GIIS that the GRIS had disappeared.

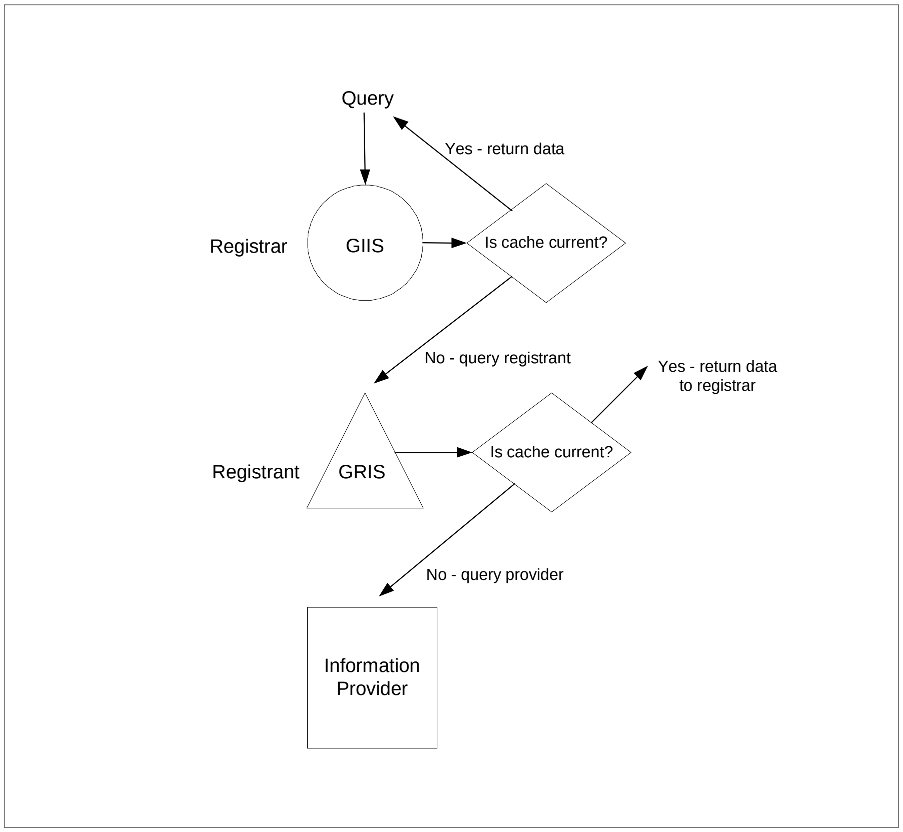
Cache ttl (cachettl)

The cachettl parameter is in the grid-info-resource-register.conf file. The cachettl specifies how long the registrar should keep the registrant's data in the registrar's cache. In other words, the cachettl is the registrant's "recommended" amount of time for the registrar to keep its data in cache. The default value of cachettl is 30 seconds.

When queries are received by the registrar within the time specified by cachettl, the data in the registrar's cache is returned. When queries are received by the registrar after the cachettl value has timed out, the registrar requests the data from the next lower level of the hierarchy.

For example, say you have a registrar GIIS with a registrant GRIS and a single information provider, and the GIIS receives a query after cachettl times out. The registrar GIIS will then request data from the registrant GRIS. If cached data is available (i.e., has not timed out) from the registrant GRIS, that data will be sent to the registrar GIIS. If cached data is not available from the registrant GRIS, the GRIS invokes the information provider, receives data from the provider, and sends the data to the registrar GIIS.

This process is illustrated in the following diagram:



Note that for simplicity, the above example and diagram use a single registrar and a single registrant with one information provider. A hierarchy can consist of multiple registrars and registrants with multiple information providers. With multiple providers, a registrant GRIS checks the cachetime of each subtree associated with each information provider.

In caching, the clock starts when a registrar receives data, regardless of when the data was generated. The data might be obsolete when it comes to the top-level GIIS, but the GIIS still returns that data.

For a discussion of performance expectations from a GRIS or GIIS query, refer to *Appendix B*.

Note that MDS uses its own caching mechanism, and does not rely on that provided by standard OpenLDAP.

Note also that the proper operation of these timing and registration parameters depends on the synchronization of the system clocks on the machines involved. If the clocks are not synchronized, registration messages may be lost, and partial data or no data may be returned.

GIIS Registration Status Checking

The status of servers that are registered to a GIIS or from which a GIIS is getting data can be checked with an option on the grid-info-search command. If several servers are registered to a GIIS and it appears that not all of them are sending data, grid-info-search with the giisregistrationstatus option can be used to check the status of those servers.

Note that the term “server” is defined here as a resource specified in MDS with a host name, port number, and suffix (usually “Mds-vo-name=site, o=Grid” or “Mds-vo-name=local, o=Grid” by default).

The command output displays registration objects that include the status type: valid, invalid, or purged. These status types are derived from the validfrom:, validto:, and keepto: parameters, which are generated from the ttl: parameter in the grid-info-resource-register.conf file.

The validfrom:, validto:, and keepto: parameters represent the timeframe during which one server keeps registration messages sent from another server. This timeframe can be illustrated as follows:

```
     validfrom:      validto:       keepto:
     ↓               ↓              ↓
time=|    VALID      |   INVALID    |   PURGED   |
```

An example of these parameters from a grid-info-site-giis.conf file is as follows:

```
Mds-validfrom: 20020522174628Z
Mds-validto: 20020522180128Z
Mds-keepto: 20020522180128Z
```

These parameters show year, month, day, and time in hours, minutes, and seconds.

The grid-info-search command with the giisregistrationstatus option displays the status of the machines one level below (i.e., the immediate children) of the GIIS being queried.

For example, the following command:

```
grid-info-search -x -b "mds-vo-name=site,o=grid" -s base
giisregistrationstatus
```

displays the status of the default server (as defined in the grid-info.conf file) and a server registered to it. Note that the –b and –s base options are required on the command for querying from the default server. The –h and –p options are required for querying from other servers. See the *grid-info-search command* section below for the full command syntax.

The command output is as follows. Registration objects are displayed for a GRIS on host1 registering to the GIIS on host1, and for a GRIS on host2 registering to the GIIS on host1.

```
version: 2

#
# filter: (objectclass=*)
# requesting: giisregistrationstatus
#

# site, Grid
dn: Mds-Vo-name=site,o=Grid
objectClass: Mds
objectClass: MdsVoOp
objectClass: MdsService
objectClass: MdsServiceLdap
Mds-Service-type: ldap
Mds-Service-hn: host2
Mds-Service-port: 2135
Mds-Service-Ldap-suffix: Mds-Vo-name=local, o=grid
Mds-Service-Ldap-sizelimit: 0
Mds-Service-Ldap-timeout: 30
Mds-Service-Ldap-cachettl: 1200
Mds-Bind-Method-servers: ANONYM-ONLY
Mds-Reg-status: VALID

# site, Grid
dn: Mds-Vo-name=site,o=Grid
objectClass: Mds
objectClass: MdsVoOp
objectClass: MdsService
objectClass: MdsServiceLdap
Mds-Service-type: ldap
Mds-Service-hn: host1
Mds-Service-port: 2135
Mds-Service-Ldap-suffix: Mds-Vo-name=local, o=grid
Mds-Service-Ldap-sizelimit: 0
Mds-Service-Ldap-timeout: 20
Mds-Service-Ldap-cachettl: 1200
Mds-Bind-Method-servers: AUTHC-ONLY
Mds-Reg-status: VALID

# search result
search: 2
result: 0 Success

# numResponses: 3
# numEntries: 2
```

Mutual Authentication in a GIIS Hierarchy

MDS 2.2 provides mutual authentication between GIIS and GRIS servers as well as between GIIS servers in a hierarchy. This feature is intended to keep rogue servers from affecting a GIIS hierarchy. Mutual authentication is performed on outbound channels, as in a server registering to the one above it in a hierarchy.

Mutual authentication is enabled via the bindmethod parameter in the grid-info-resource-register.conf file. This parameter specifies the binding method from the upper-level GIIS as one of the following:

AUTHC-ONLY: Bind only with GSI authentication.
AUTHC-FIRST: Try authentication first; if that doesn't work, use anonymous binding. Authentication is tried first with every registration attempt.
ANONYM-ONLY: Use anonymous binding only. This is the default.

Note that the bindmethod parameter is required in the grid-info-resource-register.conf file. Without this parameter in the file, registration will not work.

Mutual authentication works only for servers in an MDS 2.2 environment. That is, there cannot be mutual authentication between MDS 2.1 and 2.2 resources.

A related policy statement may be included in the grid-info-site-policy.conf file to restrict mutual authentication to a particular host or hosts and to a specific port on a host or hosts. In addition to the host/port information, the policy statement in this file would appear as:

policydata: (Mds-Bind-Method-servers=*bindmethod*)

where *bindmethod* is one of the three binding methods above. The binding method specified in grid-info-site-policy.conf should match that in grid-info-resource-register.conf.

Refer to the *grid-info-site-policy.conf* section below for more information on policy statements.

Additional Information on Site Policy, Configuration Files, and Command Syntax

This section describes additional information of interest in creating and querying a hierarchical GIIS.

grid-info-site-policy.conf

This file allows you to control the acceptance of registration messages. MDS 2.2 is designed to enable both resource discovery and monitoring. Site policy allows you to create an open policy where all registrants are welcome, or a closed system whereby only resources that you specify in the grid-info-site-policy.conf file will be able to register with a GIIS. This file is located in the GLOBUS_LOCATION/etc directory.

You can configure the grid-info-site-policy.conf file to accept registrations from anyone via a wildcard, or to accept registrations from a list of specified hosts via a Boolean expression.

We recommend that you first configure grid-info-site-policy.conf with the default policy data shown below. Each GIIS in an initial or test hierarchy should use this policy data first to verify proper operation. In this case, you are configuring a single machine to accept registrations from itself.

For all of the GIISs in an initial or test hierarchy, we then recommend that you change the default policy to the completely open policy shown below. Doing this will help eliminate any initial configuration problems. After you verify that your architecture is working with this open policy data, you can modify acceptance of registrations by substituting a policy data block appropriate to your security requirements.

The following example data blocks illustrate the use of grid-info-site-policy.conf for default, completely open, and restricted registrations. Note that the *policydata:* line in this file must be on the very next line after the *objectclass:* line; there should not be a blank or any other line between these two lines. This is a requirement of LDIF syntax standards as described in *The LDIF Data Interchange Format - Technical Specification* (RFC 2849: ftp://ftp.isi.edu/in-notes/rfc2849.txt).

Default Policy Data:

objectclass: MdsRegistrationPolicy
policydata: (&(Mds-Service-hn=*hostname*)(Mds-Service-port=2135))

where *hostname* is the name of the machine on which the GIIS is running and accepting registrations from itself, and only from port 2135. This default is intended only for a single machine.

Change 'policydata' to this for completely open policy:

(Mds-Service-hn=*)

This allows registration from any available host and from any port.

Change 'policydata' to this to restrict to 2 specific hosts:

(&(|(Mds-Service-hn=host1.isi.edu)(Mds-Service-hn=host2.isi.edu))(Mds-Service-port=2135))

This allows registration only from host1.isi.edu and host2.isi.edu, and only from port 2135. You can use a Boolean expression similar to this example to specify additional hosts or ports.

Change ‘policydata’ to this to restrict to the local GRIS:

(&(Mds-Service-hn=my-machine)(Mds-Service-port=2135)(Mds-Service-Ldap-suffix=Mds-Vo-name=local, o=grid))

This allows registration only from the local GRIS; all other registration messages are filtered out. Note that the spacing in the suffix specification (between “Mds-Vo-name=local,” and “o=grid”) must match that in the grid-info-resource-register.conf file.

Why is mds-vo-name=local used for the GRIS?

Within the grid-info-slapd.conf and grid-info-resource-register.conf configuration files, setting mds-vo-name=local specifies that it is a GRIS (versus a GIIS). In other words, MDS assumes that any value other than *local* for the mds-vo-name implies that it is receiving a GIIS. This is important because there are differences in how the data is propagated up the hierarchy depending on whether it is coming from a GRIS or a GIIS.

For a GRIS (mds-vo-name=local), the vo is renamed from *local* to the mds-vo-name of the GIIS to which it is propagating. For a GIIS, the vo's are appended (to the dn: values returned from the information provider) as they propagate up the hierarchy. See the renaming example above for details.

grid-info-search command

The syntax for the grid-info-search command is as follows:

```
grid-info-search [options]
```

where `options` are as follows:

`-config file`
Specifies a different configuration file to obtain MDS defaults.
Note that this option overrides the variables set in the user's environment and previously listed command line options.

`-mdshost host (-h)`
Is the host name on which the MDS server is running. The default is $GRID_INFO_HOST.

`-mdsport port (-p)`
Is the port number on which the MDS server is running. The default is port $GRID_INFO_PORT.

`-anonymous (-x)`
Uses anonymous binding instead of GSSAPI.

`-mdsbasedn branch-point (-b)`
Is the base location in DIT from which to start the search. The default is "${GRID_INFO_BASEDN}".

`-s` *scope*
Specifies the scope of the search. *scope* can be `base` (base dn level), `one` (base dn level plus one level down) or `sub` (all from base dn level and down).

`giisregistrationstatus`
Specifies that the status of machines registered to a GIIS or from which a GIIS is getting data be returned from the search. This option requires the `–b` and `–s base` options on the command for querying from the default server (as defined in the grid-info.conf file). The `–h` and `–p` options are required for querying from other servers.

`“attribute”`
Specifies a single attribute to be returned from the search, such as MdsCpu.

`“filter”`
Sets an attribute relationship to a value for the search:

```
“(attribute=value)”
“(attribute>=value)”
“(attribute<=value)”
“(attribute~=value)”
```

`“(attribute=*)”`	Searches for the presence of an item.
`“(attribute=*a)”`	Searches for a substring item.

`-mdstimeout seconds (-T)`
Is the amount of time (in seconds) one should allow to wait on an MDS request. The default is $GRID_INFO_TIMEOUT.

Appendix A. Registration Configuration File Reference

The grid-info-resource-register.conf file is created (like the other MDS 2.2 configuration files) as a result of the MDS 2.2 installation. This file resides in the GLOBUS_LOCATION/etc directory.

All configuration files used by MDS are described in *MDS 2.2 Configuration Files*.

The parameters defined in grid-info-resource-register.conf are as follows:

```
dn: <LDAP add object DN>
regtype: <version level>
reghn: <host to send registration to>
regport: <port to send registration to>
regperiod: <length of time between outgoing registration messages (seconds)>
[service attribute/value]...
```

Note that regtype is the version level of the MDS software. For example, any resource running MDS 2.2 would use a regtype of mdsreg2.

The [service attribute/value] entries depend on the type of LDAP object being published. For MDS 2.2 registration objects, the attributes are:

```
type: ldap
hn: <host name of registrant>
port: <port of registrant>
rootdn: <DN suffix of registrant>
ttl: <length of time to keep registration data in the registrar>
timeout: <after how long should a client abandon queries to registrant>
mode: cachedump
cachettl: <length of time for client to cache data>
bindmethod: binding method for the upper level giis
   AUTHC-ONLY/AUTHC-FIRST/ANONYM-ONLY
```

The following is an example of a grid-info-resource-register.conf file:

```
# for default MDS 2.2 install
# register this server GRIS to this server GIIS
dn: Mds-Vo-Op-name=register, Mds-Vo-name=site, o=grid
regtype: mdsreg2
reghn: dc-user.isi.edu
regport: 2135
regperiod: 600
type: ldap
hn: dc-user.isi.edu
port: 2135
rootdn: Mds-Vo-name=local, o=grid
ttl: 1200
timeout: 20
mode: cachedump
cachettl: 30
bindmethod: ANONYM-ONLY
```

Appendix B. Performance Expectations From a GRIS or GIIS Query

The performance of a query to a GRIS is dependent upon the performance of the information providers that the GRIS accesses, as well as the TTL data for the information they deliver:

- When a query to a GRIS arrives, if the data requested is live and cached, the query will be answered very quickly.

- If the data requested has been flushed from the cache because it has expired, then the GRIS will invoke the information provider or providers that supply the information required by the query. If these providers start up and deliver information quickly, then the GRIS can in turn answer the client query relatively quickly, though it will be a little slower than if the data had been in cache.

- If an information provider takes a long time to deliver its information, either because there is a lot of overhead for it to start up or acquire the data, or because there is a lot of data to deliver, then this will negatively impact the performance the client query sees.

- Further, the GRIS will allow a certain, configurable, time window for the information provider to finish its delivery. As a result, if the provider does not finish within the time window, the GRIS will terminate its link to the provider and move on to the next provider, if any. As a result, data from that provider may never appear in the GRIS. To resolve this, either extend the time window that the GRIS waits, or send less data to the GRIS. In this case, though, the client query to the GRIS will take a long time to return, since it will wait for the timeout window and the amount of time to access any other information providers before it can answer the client query.

- To extend this time window for the information provider, try increasing the value of the timelimit: parameter for the relevant provider in the etc/grid-info-resource-ldif.conf file. (Refer to *MDS 2.2 Configuration Files* for more details on and an example of this file.)

The performance of a query to a GIIS is dependent upon the performance of the GRIS's that it accesses, as well as the TTL data for the information they deliver:

- When a query to a GIIS arrives, if the data requested is live and cached, the query will be answered very quickly.

- If the data requested has been flushed from cache because it has expired, then the GIIS might query a GRIS that supplies that information. The performance of this sub-query is like any client querying a GRIS, as described above.

- Alternatively, in a GIIS hierarchy, a GIIS query for un-cached data might in turn query a subsidiary GIIS. The performance of this sub-query to another GIIS is based upon whether it has cached data, and if not, the performance of its sub-hierarchy.

In summary, there is no a priori formula for predicting the performance of a query to MDS. The more complex the GIIS hierarchy, the more unpredictable the performance, and in general, the longer that a query might take to answer, depending upon TTL data.

It is possible to mitigate this performance variability by attempting to insure that data queried is normally in cache. One can do this either by increasing the TTL value for the data (but that might not make sense), or by causing the cache to fill by periodically running a script that triggers this. This can be done at any or all levels of a GIIS hierarchy, and at the GRIS level as well. If the caches are filled regularly by scripts running periodically, then client queries will most likely find the data they are looking for in cache.