M. I. T. PROJECT ATHENA

Moira User's Manual

Mark Rosenstein
1 February 1991 at 16:31

M. I. T. PROJECT ATHENA

Moira User's Manual

Mark Rosenstein
1 February 1991 at 16:31

This is a user's and operator's manual for moira. It is primarily targeted for administrators who use moira to manage a distributed computing environment, although there are sections of interest to regular users as well. There are separate manuals to describe compiling the system, making local modifications, and the design decisions behind it.

This manual attempts to document both what moira is capable of and the conventions that we use here at Project Athena. Sections describing these conventions are marked with a [Athena] symbol and may be done differently at other sites.

1. Moira Overview

Moira is the Project Athena Service Management System. It manages the configuration of all of the Athena network services. It consists of a large relational database, and front end software to control access to that information and automatically update system servers from that information.

Moira was developed at MIT's Project Athena, and has been in daily use here since 1988. It is available with the following restrictions:

Moira depends on the Kerberos authentication system, and will make use of the Hesiod nameservice and Zephyr notification services if they are present in your system.

There are a number of client programs available, which can be run on any workstation. These include

- moira (listmaint, usermaint, and dcmmaint are part of this)
- mrcheck
- mrtest
- blanche
- chfn
- chpobox
- mailmaint

The text in this document assumes that you are using the **moira** client program unless otherwise indicated. References are also made at various points to the actual moira queries. These are listed in a separate document, *Moira Queries*. Queries may be used with the **mrtest** program.

Moira can be used by everyone on the system, but some operations require specific privileges. Different parts of moira are of interest to different users. Regular users will probably not be aware of what moira is, yet will use it when they run **chfn** or **mailmaint**. More experienced users may do more list manipulation, using **listmaint** and **blanche**. Only users with privileges will be able to use much more than that.

[Athena] Here, we have three kinds of advanced moira users: the user accounts consultants, the operations staff, and the moira manager. The user accounts consultants handle problems and questions from users about their accounts, and are the people who do most of the manipulation of accounts and lists. The operations staff handle configuration of machines, clusters, printers, and fileservers. Finally, there is one person designated as the moira system manager who sees that the system runs smoothly and has occasional maintenance to do.

1.1. Moira Objects

There are several kinds of first-class objects in moira: users, lists, machines, clusters, and filesystems. Other objects in the database are not as flexible. These first-class objects share some characteristics.

Each of these objects has a name, and while all names of a given type are unique (for example: no two users have the same login name), two different kinds of object may have the same name (a user may have the same name as a list). Names must consist of printable characters excluding double quotes, asterisk, question mark, backslash, and square brackets (" * ? \ [ ]).

These objects are often referenced by other objects in the database. These references are done in such a way that if you change a user's login name, all references to that user will now show the new name. Beware that some other objects in the database do not share this characteristic: printers, DCM services, and zephyr ACLs.

Both the first-class objects and many other objects in moira all record last modification info. This includes the date and time it was modified, the user responsible for the modification, and the program that was used. Some changes affect the modification info on related objects. For example: changing the information associated with a cluster will update the cluster modification time as well.

2. Regular Users

Regular users only have permission to examine their own account, change some of their account information, and manipulate public mailing lists and lists that they own. There are a few more operations that anyone may perform, but most users will not find too useful.

2.1. Name, Address, Phone Number

The **chfn** program is used to set how your full name looks to other users, and your address and phone number as seen by finger. When started, **chfn** will tell you when your info was last changed and by whom. If you just press *return*, **chfn** will reuse the old value in that question. To make an answer empty, type ‘‘none’’ (without the quote marks). Note that you cannot have a colon ‘‘:’’ or a comma ‘‘,’’ in any of your answers.

Your full name can be whatever you want it to read. The operations staff has a separate record of who you really are, so you cannot hide from them. Your address and phone number will be accessible to people who finger you if you enter them here. You can also keep them blank.

When you change this information, the changes will not take effect until the next update. [Athena] This will happen by the next morning at the latest. If you are a system administrator, you can change other people's finger info by running ''chfn *username*''. **Chfn** uses the queries `get_finger_by_login` and `update_finger_by_login`.

2.2. Receiving Mail

[Athena] Most people receive their email in a post office box, or pobox, which refers to a special server machine that Athena uses. If you want email sent to your account to go somewhere else, you can use the **chpobox** program to change this. Note that at a site using POP servers like Athena, a *.forward* file in your home directory will have no effect.

To see your current pobox, type

```
chpobox
```

To forward your mail to to another site, use

```
chpobox -s user@host.domain
```

To undo the above forwarding and receive your mail locally in the same pobox you used before, type

```
chpobox -p
```

Note that any change requested by **chpobox** will not take effect for a while, [Athena] but should be in place by the following morning at the latest.

Note that moira will canonicalize the hostname if you request mail forwarding. In most cases this is correct, but you may want to override this. For example, you may want to forward your mail to ''user@LCS.MIT.EDU'', where LCS.MIT.EDU is an alias for the machine PTT.LCS.MIT.EDU. In this case you may not want the name LCS.MIT.EDU to be canonicalized, since in the future the alias may point somewhere else. In this case, you could use the command

```
chpobox -s user@\"lcs.mit.edu\"
```

to avoid having the hostname canonicalized. Only the double-quotes are necessary for the **chpobox** program; the backslashes are there to get the quotes past the shell.

Sometimes a user really wants mail to go more than one place without using a mailing list. This is not recommended, but can be done by setting the pobox to an external address which is actually a comma separated list, like this:

```
chpobox -s "user@host.domain, user@another.host.domain"
```

[Athena] Note that if one of the destinations is a regular pobox, it is not acceptable to use ''user@ATHENA.MIT.EDU'', you must instead use an address like ''user@ATHENA-PO-1.LOCAL'' when specifying multiple poboxes. The user will also have to set the environment variable MAILHOST to the name of their POP server if they do this, since moira no longer recognizes that they have a POP box.

Each of the above commands also takes an additional argument ''-u *username*'' that an administrator can use to modify someone else's mail forwarding. There is also a menu choice in the **moira** program for manipulating poboxes. The queries pertaining to poboxes are `get_pobox`, `set_pobox`, `set_pobox_pop`, and `delete_pobox`.

2.3. Mailing Lists

This section gives a brief description of manipulating lists. For a more complete treatment of creating lists, their attributes, and other options on them see section 3.2.

Any user may add or delete their own usernames from public mailing lists. This is easiest done with the **mailmaint** program. **Mailmaint** presents you with a menu containing the following items:

1. Show all mailing lists.
2. Get all members of a mailing list.
3. Display lists of which you are a member.
4. Show description of list.
5. Add yourself to a mailing list.
6. Delete yourself from a mailing list.

The first choice will list all of the public mailing lists, but not private ones. This may take a while, as there are liable to be many mailing lists to choose from.

Choice two will display all of the members of a list. You will only be able to do this for lists which are visible (i.e. not *hidden*) unless you are an administrator of the list or you have privileges within moira. Lists can contain several kinds of members. Those you are most likely to see are users, which name users of the local system, and strings, which are external mail addresses. Another list may be a member of a list as well, in which case every member of that sub-list is also a member of this list.

To find out which lists you belong to, you can use choice three. Note that you may be a member of some groups in addition to mailing lists, and these will be displayed at the same time. To find out more about any list, you can select option four.

Finally, choices five and six allow you to put your username on a list or take yourself off. Note that you can only do this to lists that are public or that you administer, so you may get a ‘‘permission denied’’ error. In this case, you will have to ask either the administrator of that specific list, or a user accounts consultant to make the change.

2.3.1. For More Sophisticated Users

If you manage a list, or want to do more than the **mailmaint** program allows, there are two other programs that you may find useful. **Listmaint** is a section of **moira**, and will allow you to change the attributes of a list, change the membership of a list (one member at a time), and find out more about lists. **Blanche** is a program that is faster for quick checks of a list or making many membership changes to a list. See section 3.2.2 for more information.

3. User Accounts Administrator

[Athena] The accounts administrators are primarily responsible for user accounts, poboxes and mailing lists, and adjusting user quotas. Quotas will be covered later along with other aspects of filesystems.

3.1. User Accounts

Moira stores a lot of information about users in its database. It should have information about every potential user in your community, whether or not they actually have an account. For each user, moira knows:

login — Their login name if they have one. This may be up to 8 characters long. For someone with status 0 or status 4 (not registered or not registerable), it is usually a hash-mark followed by their UID (i.e. #17822), but the value at this point really doesn’t matter. Moira will not let two accounts have the same login name. A login name may be set to the string ‘‘create unique login

ID'' in which case moira will set the login name to something guaranteed to be unique (a hash sign followed by a unique number).

UID — Their UNIX user's ID. These are stored as 16-bit signed integers. This may be set to a numeric value, or the UID may also be set to the string ''create unique UID'' in which case moira will assign one automatically that is not already in use within moira. The software that assigns them automatically will put them in the range 100-32765.

shell — Their login shell. To moira this is just a string, up to 32 characters long. It is used to construct the password entry.

name — Their real first, middle, and last names where they can't be changed by the user. Each part may be up to 16 characters long.

status — Their account status. See the chart on page 6.

class — Their class. To moira this is just a string, up to 8 characters long and case insensitive. [Athena] At Athena we use it to store the reason this account exists. See the following table for a list of classes in use at Athena.

encrypted ID number
This is the user's school or organization ID number, passed through a one-way encryption. We use the UNIX password algorithm, using the first and last initials of the user's real name as the salt. The code expects the ID number to be 9 digits.

[Athena] The classes in use at Athena are:

1992, 1993, etc. — An undergraduate, indicating the expected year of graduation. We will not turn off an account before June of the indicated year.

G — A grad student. They are here for varying lengths of time, and take leaves-of-absences as well. We use the latest records from the registrar to determine who is eligible.

STAFF — These are just Project Athena staff members.

MITS — MIT Staff other than those working for Project Athena.

FACULTY — MIT Faculty members. This includes anyone involved in the teaching of a course who will need access to Athena.

PROJECT — Special accounts for use by projects. We are trying to phase these out.

SYSTEM — Special accounts for use by system software. This includes the obvious ones like ''root'', ''daemon'' and ''uucp'', and ones needed by software packages like ''rtingres''.

FALL90, SPRING91, etc
Special students and others needing an account for just a short while. The name lets us know when we can turn off these accounts.

GUEST — Guest accounts.

TEST — Accounts used for testing either workstations or moira and the registration process. Most of these may be deleted at any time.

COURSE — Like project accounts, we are trying to phase these out.

OTHER — Accounts that don't fall into any other category.

Special groups: SIPB, MITES, ESP, CAES, KNIGHT, HST, WHOI, INTERPH
These are organizations that have people who aren't officially part of the MIT community who have Athena accounts.

The account status field identifies if an account is active, among other things. [Athena] At Athena, we have two ways an account might be active: as a full Athena account with access to Athena resources, and simply as a user in the campus namespace. A user of the campus namespace may be on mailing lists, get mail forwarded through the mailhub, and have a kerberos principal whose name is unique among Athena login names, but may not actually use Athena workstations. We use the verb ''register'' to refer to getting a real Athena account, and ''enroll'' to being entered in the campus namespace. The integer values in the status field for a user mean:

0 - Registerable
: This is the way students, faculty, and others eligible for accounts are first entered in the database.

1 - Active
: This is a completely active, normal account.

2 - Halfway registered
: This is an intermediate step between states 0 and 1 used by the reg_svr program.

3 - Marked for deletion
: This account was once active, but has been turned off. It will really be deleted at some point in the future. While it is in this state, no network services will know about it.

4 - Not allowed to register
: This is how people are added to the database who are part of the community but not eligible for accounts.

5 - Enrolled in campus namespace
: This is someone who has enrolled in the campus namespace, and is still eligible to register for an account as well.

6 - Enrolled, not allowed to register
: This is someone who has enrolled, and is not allowed to register for an account.

7 - Half enrolled in campus namespace
: This is an intermediate step between states 4 and 6 used by the reg_svr program.

3.1.1. Looking Up Users

There are several ways to look up existing users. Using the **moira** client, users may be retrieved by login name, real name, or class. When looking someone up by their name, try to enter their entire first name or last name. If you use wildcards in both parts of their name, the lookup will be much less efficient. Beware of doing a lookup by class, and this may return many records. Queries that return more than a couple hundred records can cause the server to crash. When using **mrtest**, it is also possible to lookup a user by their UID or by their encrypted school ID, using the queries `get_user_by_uid` or `get_user_by_mitid`.

3.1.2. Adding New Users

When users are added by hand, three items are necessary: the person’s real name, their ID number, and their class. Most of the time, the login name, UID, shell, and status should be allowed to default to ‘‘create unique login ID’’, ‘‘create unique UID’’, ‘‘/bin/csh’’, and 0 respectively. The moira client will also look up the name in the database to see if that user already appears there.

The name must be correct for accountability. It generally appears exactly as the person is known by the school administration, which may be somewhat different from the way the person likes to print their name. There is another field where the user can specify how they want their name to appear to other users.

Moira is case-sensitive on names. [Athena] The school administration sends them to us in all uppercase from their mainframes, but in an effort to have them look better, we apply the following algorithm: the first letter of the name and any letter following whitespace or punctuation will be capitalized, the rest being lowercase. This works correctly for names like ‘‘O’Connor’’, but will be wrong for names like ‘‘McIntosh’’ or ‘‘van Buren’’. Names entered by hand should be entered capitalized according to this algorithm or the user will not be able to user the register program. Moira also strips off any suffixes like ‘‘Jr’’, or ‘‘III’’.

An ID number must be entered. If one is made up, rather than entering the correct ID number, then a second entry will be made for this person if we receive information about them on a tape from the registrar or personnel office. If this person is a guest or someone who will not appear on one of these tapes, make up a number for them. It must be 9 digits long.

The class must be entered correctly to make it possible to know when this account is no longer needed and may be deleted. A scheme should be adopted for how this field is used if it is to be useful.

3.1.3. Making a User Account

Once a user is in the database, they may claim an account by running the register program, or by an administrator registering them using the register option of the user menu in **moira**. Even if an administrator is going to create the account, we suggest first entering them as above and then registering the account, rather than entering them as an active account to begin with. This makes sure that no part of the account is forgotten. If you are creating a system account and want to do it by hand, enter the fields as follows.

The login name must be unique among all login names known to moira. The UID may be left as ‘‘create unique UID’’, or specified as any 16-bit number. It may duplicate an existing UID. The shell should be set to whatever value you want appearing in the /etc/passwd file for the user’s shell. The status may be immediately set to 1. This will create an account without a filesystem, pobox, group list, quota, or kerberos principal.

If a user account is owned by someone who often works in a different kerberos realm, it is possible to store kerberos mappings in moira so that a foreign kerberos principal (or another instance in the local realm) may have the privileges of a user in moira. This is done in the *krbmap* sub-menu. Each kerberos principal must map to exactly one user. The principal is specified in a case-sensitive string like ‘‘user.instance@REALM’’ or ‘‘user@REALM’’ if the instance is null. Note that any periods in the username or at-signs before the realm name must be escaped by a backslash.

3.1.4. Changing User Information

Any of the user information may be changed. If the login name is changed, the new name must be unique among all login names known to moira. All references to that user within moira will now refer to the new login name. However, it is still necessary to change the filesystem name and group name to change all of the information associated with the user.

The class field will be automatically changed by the programs that process tapes of information from the registrar and personnel offices if that user’s status with the school changes.

3.1.5. Removing Users

In general, users are only deactivated by changing their status to 3. They are not actually deleted in the course of day-to-day maintenance by the accounts consultants. These deletions occur in batch once or twice a year, see page 19. In fact, the **moira** client will refuse to delete any account that is not in status 0.

If you really do want to delete a record of a user account, you may do so if it has never been registered (i.e. still in state 0) or if you first change it to state 0. Any quotas the user has assigned or kerberos mappings will automatically be deleted. The user record must not be in use in any of the following ways:

- must not be a member of any lists
- must not be the owner of any filesystems
- must not be the owner of any lists

- must not be the owner of any services
- must not be on a hostaccess list

The **moira** client will prompt you to delete an existing filesystem with the same name as the user assuming that it is the user's home directory. It will also prompt you to delete a list with the same name as the user's group, and to remove the user from any lists that the user may be a member of.

3.2. Lists

Lists in moira serve three purposes: they can be mailing lists, UNIX groups, or access control lists. Any list can have any combination of these uses. Lists may contain users, other lists, character strings (i.e. foreign mail addresses), or kerberos principals. When a list contains another list, it is as though all of the members of the sub-list are also members of the parent list, and this is handled recursively for as many levels as necessary. There is a limit within the current implementation of the server where the tree structure used to compute intermediate list memberships cannot have more than 1000 entries, but this is a constant that can be raised by recompiling and has not proven to be a problem in practice.

We somewhat haphazardly use the terms list administrator, list owner, list ACL, and list ACE to all mean the same thing. The correct term is ACE for Access Control Entity, although administrator is preferred for readability.

Lists have the following attributes:

name	This can be up to 32 characters. To avoid mail problems with the names of mailing lists we recommend that the names be all lowercase and do not start with a numeral or contain spaces or other punctuation characters such as comma, period, at-sign, semicolon, parenthesis, or angle brackets (, . @ ; ( ) < >). List names must also be unique within the moira database.
active/inactive	This is a flag indicating whether this list should be present in updates to the system services. When a list is made inactive, it is as though it does not exist, except that it is easy to reinstate by just toggling this flag again. This is how we make groups disappear that belong to accounts that are marked for deletion.
public/private	This flag indicates if a list is public, meaning than anyone can add their own username to the list or delete their username from it. On the other hand, if a list is private only an administrator of the list or a moira administrator can change it.
visible/hidden	This flag indicates if a list is visible, in which case anyone can see the list attributes and membership. When a list is hidden, moira will only let the list administrator or moira administrator see the list attributes or membership. Note that even if a list is hidden, the mailhub will tell people the list membership.
maillist	This is a flag that indicates if this list should be treated as a mailing list and downloaded to the mailhub. Some lists may appear on the mailhub that are not specifically marked as maillists, since any list that is itself a member of another list which is a maillist will be treated as a maillist. Also, if a list is an administrator of a maillist, it will be treated as a maillist (so that mailer errors pertaining to the list can be sent to the administrators).
group and GID	Group is a flag indicating if this list should be treated as a UNIX group or not. If group is true, then GID must be a unique 16-bit number. The GID may be entered as a number, , or set to the string ‘‘create unique GID’’ in which case moira will assign one automatically that is not already in use within the database. The software that assigns them automatically will put them in the range 100-32765.
description	This is a simple description of the list, and may be up to 255 characters long and may contain whatever you want.
ACE	ACE stands for Access Control Entity. This is like an access control list, except that it is not necessarily a list. It consists of two parts: a type, and a name. The type is one of USER, LIST, KERBEROS, or NONE. If the type is USER, the name is a login name known to moira. If the type is LIST, the name is a list name known to moira, and may be self-referential (i.e. the list is

its own administrator). If the type is KERBEROS, the name is a kerberos principal which is case-sensitive in the form ‘‘user.instance@REALM’’ or ‘‘user@REALM’’. If the type is NONE, then any value may be passed as the name and it will be ignored.

3.2.1. Creating & Modifying Lists

Lists may be created with **moira** or **listmaint**. You will be asked for each item described above. If the list is not to be a group, then the GID is automatically set to ‘‘create unique GID’’.

[Athena] When a list is created to contain just a few users to be the administrator of another list, the convention is to call it *list*-acl, where *list* is the name of the list it will administer. If this list is to be an Internet mailing list, you can kill two birds with one stone by calling the owning list *list*-request, which is the Internet-wide convention for contacting mailing list administrators. If you must create both a list and its administering list, remember to create the administering list first, since moira will not let you specify the name of a not-yet-created list as owner. Most administrator lists are self-administering, so that any administrator can grant or revoke that privilege to other administrators.

The administrators of a list can change any of its attributes except the GID, which can only be changed by moira administrators.

A list may be deleted unless it

- has any members
- is itself a member of any lists
- is a filesystem owner
- is in use as a query ACL
- is a list owner
- is a service ACL
- has a group quota
- is a hostaccess ACL
- is a zephyr ACL

The **moira** client will attempt to remove the members of a list and remove the list as a member of any other lists while deleting a list. It will display how else the list is in use if it still cannot delete the list.

3.2.2. Membership

List members may be of type USER, LIST, STRING, or KERBEROS. A member of type USER is identified by login name, and must be an account known to moira. If the account’s login name is changed, this change will automatically be reflected in the list membership. If a user is a member of a maillist, then mail will be sent to the user’s pobox. A user member of a group simply puts that user in the group. A user member of a moira access control list means that someone authenticating to that user or a kerberos principal mapped to that user has the privileges granted by the access control list.

A member of type LIST is identified by the list name, and must name a list known to moira. If the list’s name is changed, this change will automatically be reflected in the list membership. When a list is a member of a maillist, the sub-list will also be treated as a maillist and mail sent to that maillist will also be sent to all members of the sub-list. When a list is a member of a group, all members of that sub-list are also members of the group.

A member of type STRING is identified by the string itself. Strings may contain spaces and other punctuation, but beware of asterisk, question mark and square braces (* ? [ ]) which are special characters within the database. You may store them normally, but to retrieve based on a string containing those characters, you must precede them by backslashes to avoid their taking on pattern-matching meanings. A string member of a group or access control list is ignored. A string member of a mailing list is assumed to be an external mail address.

A member of type KERBEROS is identified by a text-representation of the principal-instance-realm triple. The principal is specified in a case-sensitive string like ‘‘user.instance@REALM’’ or ‘‘user@REALM’’ if the instance is null. Kerberos members of maillists are ignored. Kerberos members of groups are ignored for NFS servers and UNIX groups, but propagated to the AFS protection database. KERBEROS members on moira internal access control lists apply to users who authenticate as that kerberos principal and do not map to a user.

List membership may be changed with **mailmaint**, **listmaint** (**moira**), or **blanche**. **Mailmaint** is intended for naive users. **Listmaint** is the **list** menu of **moira** and is the most complete tool for manipulating lists. However, it can be tedious to use **listmaint** to make a lot of membership changes, in which case **blanche** is a good choice. **Blanche** will allow you to read a list of users from a file and put them on a list, or take a number of changes from the UNIX command line.

3.3. Mail Forwarding

The user accounts consultants will also have to deal with mail forwarding. Setting poboxes is covered in section 2.2, although the *pobox* menu of **moira** may be used as well.

There are two possible problems to note, both of which depend on bugs/features of sendmail (these are in sendmail version 5.61, but have been there for a while). If there is at least one bad address on a mailing list, which includes putting a user on a mailing list who doesn’t have a pobox, then sendmail will just give an error for the entire list without delivering it to anyone. If a user has a pobox and there is a mailing list with the same name, the user’s pobox will take precedence over the mailing list. Moira outputs both in the aliases file, and sendmail only pays attention to the latest definition of an address.

4. Operations Staff

[Athena] The operations staff is responsible for workstations, servers, managing disk space (with the help of the accounts administrators), and printers. The following sections cover how moira interacts with these.

4.1. Managing Workstation Information

For each workstation, moira knows the name and type. The name is the fully qualified domain name. For instance, ‘‘ATHENA.MIT.EDU’’ is a valid name, but ‘‘ATHENA’’ is not. Machine names are *not* case sensitive as they are all converted to uppercase when the moira server gets them. Note that the **moira** user interface automatically canonicalizes any hostnames typed using the domain name system. If you want to enter a name and not have it canonicalized, enter the name surrounded by double-quotes ("). Each machine has a type as well. These are just strings to moira, although they are type-checked and case in-sensitive. [Athena] We use the types VAX, RT, PMAX, PS2, MAC, RIOS, NEXT, and IBM (mainframes).

A cluster is a grouping of machines with their assigned resources such as file servers, printers, etc. Each cluster has a name, a description, and a location. The location is actually just an additional description field for the users’ convenience. [Athena] We use the cluster naming convention

```
[building name][optional designation]-[architecture]
```

For example, m11-vs, e40test-rt, w20a-vs (the ‘‘a’’ designation because there are multiple vax clusters in building W20), m66priv-rt (this has *priv*ate workstations only in it).

Each cluster has information associated with it. Cluster information consists of a label and a text string. [Athena] Labels in use at Athena include ‘‘syslib’’ and a filesystem name for the system pack containing OS binaries, ‘‘usrlib’’ and a filesystem name for a secondary system pack, ‘‘lpr’’ and a printer name to be the default printer for the workstation, ‘‘zephyr’’ and a hostname to specify the preferred zephyr server for the workstation. [Athena] The Athena convention for system pack names is that each pack or filesystem has one name, and if several clusters in adjacent buildings all use one pack, they each refer to this pack by its name. This is clearer than having multiple filesystem names that point to the same actual bits on disk.

Each workstation may be in any number of clusters. If a workstation is not in any clusters, it will have no cluster information. If it is in one cluster, its info is that cluster’s info. If a workstation is in more than one cluster, its cluster info is the union of the info for each of the clusters it is in. This is useful as you may have workstations of two different architectures in one room; the machines of each architecture need different system packs, but the same default printer. You can use 3 clusters: two each identifying system packs for the two architectures, and one identifying the printer for both. This is better than putting the printer information in with the architecture specific cluster since there is only one place to change if the printer configuration is changed.

4.1.1. Adding a New Cluster

When adding a new cluster, first check to see that a new cluster is really needed. Machines in a new building that will use existing fileservers in another building may be put in the other building’s cluster, although creating a separate cluster may be clearer and will be required if a different printer is needed. Be sure to write something informative into the description and location fields if they are not obvious from the name.

Next you will need to add the appropriate information for the cluster. At a minimum this would be the syslib information, and may require usrlib as well. If there is one printer that is obviously closest to machines to be placed in this cluster, then add an lpr entry for the printer as well.

4.1.2. Adding a Machine

Is this really a new machine, or are you really trying to rename an existing machine? If this is a rename, see section 4.1.4.1 below.

Determine the machine architecture, and enter the name and type (architecture). Note that if you type a nickname instead of the primary name of the machine, **moira** will substitute the canonical name when it enters it. Next you will need to add the machine to one or more clusters. Determine the appropriate cluster(s) based on location and architecture, and add the machine.

4.1.3. Changing Cluster Information

The cluster name, location or description can easily be changed. If the name is changed, it will automatically be updated in the machine to cluster mappings as well. The cluster information may be changed by adding or deleting cluster information. It is possible for the same label to appear more than once in a cluster’s info.

4.1.4. Changing Machine Information

If a workstation is replaced by another of a different architecture, all that is required is to change the machine type. If a workstation is moved from one place to another, it will probably need to be removed from one cluster and added to another. There are several cases for what to do when the name is changed.

4.1.4.1. Changing the name of a machine

If the change has not yet taken place in the Domain Name System, you can simply update the name in the database, being sure to type the full domain name for the new name. If the DNS has already been changed, and the old name is kept as a cname (nickname) in the DNS, you will have to put the old name in double-quotes (") when typing it or the **moira** client will canonicalize it to the new name before looking it up in the database. Note that the sequence of events required so that the workstation remains functional throughout the name change are:

1. Change the DNS.
2. Update moira.
3. Change the name on the workstation itself after it is activated and before the next moira update. If the workstation is allowed to deactivate now, it will fail to activate until the next moira update.
4. Moira update of Hesiod servers occurs.

If the old name is not kept as a nickname in the DNS, then no double-quotes are necessary (although they can't hurt) when updating moira, although the workstation may not be entirely functional between the DNS update and the completion of the rename.

4.1.5. Deleting a Cluster

You cannot delete a cluster if it has any associated cluster information. First delete the cluster info, using wildcards for the label. You may then delete the cluster. If there are any machines still assigned to the cluster, moira will prompt you to delete them.

4.1.6. Deleting a Machine

You cannot delete a machine that is in use in any way other than having a cluster assignment. If the machine is in a cluster, **moira** will prompt you to remove it. If the machine is in use in any of these ways:

- has poboxes assigned to it
- receives moira updates
- has any NFS physical partitions
- spools for any printers
- spools for any palladium printers
- is a printer quotaserver
- has filesystems on it

you will not be allowed to delete it.

4.2. Managing Filesystem Information

A filesystem in Moira is an abstraction that can refer to an NFS mountpoint, an RVD pack, an AFS volume, an error indication, or a combination of these. Filesystems are often referred to as lockers so that they are not confused with the partitions that NFS lockers reside on.

Moira manages the NFS partitions as well, keeping track of quota allocation and actually creating the base directory of a locker when necessary. Moira also manages quotas that apply to both NFS and AFS lockers.

4.2.1. Lockers

Moira has the following fields for each filesystem:

name — This is a label for the filesystem, up to 32 characters long, same restrictions as list names. Each filesystem name must be unique.

type — The is the protocol type of the filesystem: AFS, NFS, RVD, ERR, or FSGROUP.

machine — This is the server that the filesystem resides on. This machine must be entered in moira. This value is only used for NFS and RVD filesystems.

packname — The name of the filesystem relative to the server. For NFS, it is the path on the server to the root of the locker. For RVD, it is the pack name. For AFS, it is the path to the root of the locker in the AFS cell. It is ignored for ERR and FSGROUP types.

default access — This is the mode that is used when the locker is attached. Typical values are ‘‘w’’ for read/write, ‘‘r’’ for read-only, ‘‘n’’ for read-only without authentication, and ‘‘x’’ for exclusive.

default mountpoint — This is where the locker is attached to the client workstation. [Athena] This is usually /mit/*lockername*.

comments — Any descriptive comments up to 64 characters. For type ERR filesystems, this is the error message.

ownership — These two fields called ‘‘owner’’ and ‘‘owners’’ identify the user and group ownership of the locker. The owner must be a user known to moira. The owners group must be a list known to moira. When the locker is created, this ownership will be used for the directory.

auto create — This flag indicates that moira should attempt to create the specified locker. Once the locker has been created, this flag may be turned off, although moira will not do that itself. Clearing the autocreate flag for existing lockers makes the NFS updates run slightly faster. Note that autocreate is currently only honored for NFS filesystems.

locker type — This field indicates what the locker is used for. This is just a string to moira. [Athena] Values in use are HOMEDIR for user home directories, SYSTEM for system software delivery, PROJECT for development project lockers, COURSE for educational course software, ACTIVITY for student activities, EXTERN for pointers to filesystems outside of Athena, and OTHER for whatever doesn’t fall into these categories.

4.2.2. NFS Partitions

Moira keeps information about the actual disk partitions that NFS servers put the NFS lockers on. For these the following info is kept:

machine — This is the host that the partition resides on, and must be known to moira.

device — The is the device name for the partition, such as ‘‘/dev/ra3e’’.

directory — This is the mountpoint of the partition. [Athena] We sometimes make this a subdirectory of the partition if we really want all lockers on that partition to be in that subdirectory. For instance, on a partition called ‘‘/u1’’ we might store the directory in moira as ‘‘/u1/lockers’’ so that all lockers would be created under that point.

status — This field actually bit-encodes two things: the intended usage of the partition and whether group quotas are in use. There are use bits for: student, faculty, staff, and miscellaneous. These use bits are simply informational except for the student bit: this bit must be set on any partition that the registration server will automatically put home directories on. The group quotas bit indicates what kind of quota extract should be done for this partition.

allocated — This field contains the total of all of the quotas of lockers on this partition. It is automatically maintained by moira when quotas are changed.

size — This should be the size of the partition in 1K blocks, although in practice this number is often changed to adjust allocations. The registration server uses this number to decide where to allocate home directories.

NFS partitions may be created on any machine known to moira, even a machine which is not receiving NFS updates. If it is not receiving updates, the directories will not get automatically created and quotas will not get set by moira.

4.2.3. Adding a Filesystem

A filesystem may be added by entering the information described above. When creating a filesystem, don't forget to assign any necessary quotas.

For type NFS filesystems, all of the fields are used. The *machine* must be known to moira. The *packname* must exist on a known NFS partition. The **moira** client will prompt you to create the NFS physical partition if it does not already exist, then retry the addition of the filesystem. The *ownership* will be used when creating the directory, if the *autocreate* flag is set.

For type RVD filesystems, the only requirements are that the *machine* the pack is on must be known to moira. All fields except *ownership* and *autocreate* will be used. Moira does not actually update RVD servers, so this information only feeds the nameservice.

For type AFS filesystems, the *machine* field is not used and is usually set to ''[NONE]'' (note that backslashes are needed before the braces when passing this string to the server to avoid interpretation as pattern matching characters). The *ownership* and *autocreate* fields are not used either.

For type ERR filesystems, only the *name, type*, and *description* are used. The remainder of the fields will be ignored. Since often the ERR type is used to temporarily deny access to a filesystem, you can change the *type* to ERR and the *description* to why access is being denied, and easily change it back later without losing the rest of the information.

File system groups are used to associate several filesystems with one name. Usually these are redundant copies of the same data for the sake of reliability, or because not all clients support all filesystem types. The filesystems that make up the group are sorted in a known order, and **attach** will try them in that order until one is successful.

In the main entry for an FSGROUP, only the *name, type,* and *description* are used. Each of the member filesystems must have different names from each other and the FSGROUP. [Athena] A useful convention is to have fsgroup *foo* containing *foo*-primary and *foo*-secondary if the group is for redundancy, or *foo*-afs and *foo*-nfs if the group is for handling multiple protocol types.

Moira also has the concept of filesystem aliases. An alias just associates another name with an existing filesystem. Because they can be confusing, their use is discouraged. [Athena] We use them primarily for developers' and testers' system packs.

4.2.4. Changing a Filesystem

If you change the name of a filesystem, references such as its membership in an FSGROUP or quota assignments will follow the new name.

If the change involves moving the files, moira will not do all of the work. This includes moving from one partition to another, moving from one server type to another, or even just changing the packname on the server. The easiest way to deal with this is to rename the existing filesystem to *foo*-old, and create a new filesystem *foo* where you want the files to end up. Then you can attach both filesystems, copy the files, and delete the old files and filesystem entry when you are sure the new one is stable.

For faster turnaround on filesystem moves, you can move the files and update moira at the same time, in which case answers from the nameserver will be inconsistent with the fileservers until the next moira update.

When a filesystem is moved between like servers, the quotas will automatically move with it. If it is moved from an NFS server with user quotas to AFS, an AFS quota will be assigned which is the sum of all of the user quotas assigned to the locker. Note that if it is moved to or from an NFS partition with group quotas that the quotas will not automatically follow. Old quotas are left in the system, so if a filesystem is later returned, the old quotas will be used again.

Access may be temporarily denied to a filesystem by changing its *type* to ERR, and putting a helpful message in the *description* field. Nothing else needs to be changed, so it is easy to restore access by restoring the *type* to its previous value.

4.2.5. Deleting a Filesystem

When deleting a filesystem, any quotas assigned to it will automatically be deleted. Do not forget to actually delete the files on the fileserver as well, since moira will not do this for you.

4.2.6. Quotas

Moira keeps track of disk quotas for multiple types of entities (users and groups) and for multiple types of filesystems (NFS and AFS). The filesystem info is:

filesystem	Which filesystem this quota applies to.
type	Type of entity this quotas applies to, currently USER, GROUP, or ANY.
name	Name of entity. This is a login name for type USER quotas, a list name for GROUP quotas, and ignored for ANY quotas.
quota	The quota value in 1K disk blocks.

If the named filesystem is of type AFS, only quotas of type ANY will be used and others will be ignored. If the named filesystem is of type NFS, the type may be either USER or GROUP. Moira determines which NFS physical partition the filesystem resides on, and the flag in the NFS physical information will indicate if this partition is a user or group quota partition. Any quotas of the wrong type will be ignored. Moira will sum up all quotas pertaining to a user on a partition and tell the server the total for each partition.

When filesystems are moved, the quota information moves with them. If a filesystem is changed from type NFS to type AFS, and a quota of type ANY does not exist for that filesystem, moira will create such a quota containing the sum of all of the USER quotas on that filesystem.

4.3. Managing Printer Information

Moira manages printcap information as well. For each printer, moira records:

name	up to 16 characters long.
spooling machine	This must name a machine known to moira. For the ‘‘rm’’ printcap field.
spool directory	Any text string (up to 32 characters) as far as moira is concerned, it will be put in the ‘‘sd’’

printcap field. This is typically /usr/spool/printer/*name*.

remote printer name
: Name of the printer relative to the spooling machine, for the ‘‘rp’’ printcap field. It is usually the same as the printer name.

quota server
: This must name a machine known to moira. This is for an Athena addition to lpr for authenticated printing. It is put in the ‘‘rq’’ printcap field.

authorization
: This flag indicates if kerberos authorization is required for this printer. It goes in the ‘‘ka’’ printcap field.

price per page
: Price in cents per page printed on this printer, to appear in the ‘‘pc’’ printcap field.

comments
: Up to 64 characters of comments

Since printers are not referred to by other parts of moira, they may be added, changed, and deleted at will. Note that moira does not support an atomic action to change a printer, so it is possible under some circumstances for a printer to accidently be deleted if an error occurs while changing it.

5. Moira System Manager

5.1. The Server Process

The heart of Moira is the process **moirad**. This must be running for Moira to be up and working.

5.1.1. Starting Moirad

If the machine crashed, or you are unsure of the state of the database, it should be checked before starting the server. Su to user ‘‘rtingres’’, and run

```
/usr/rtingres/bin/restoredb -s -f
```

to verify the Ingres database structure. It is not necessary to check the Moira database structure too often, but this can be done by running **dbck**.

Moirad is normally started using **startmoira**. It is not necessary to do this, it is a convenience program to avoid having to always have the proper environment to start the moira server. **Startmoira** will connect to the root directory, then run **moirad** capturing the logging information, timestamping it, and writing it to */moira/moira.log*. Because the **moirad** will grow in size as it runs, you may want to do an `unlimit` at the shell before starting it.

5.1.2. Shutting Down the Server

There are two different ways to shutdown moira. You can close the database to the server and leave it running, or you can actually bring down the server.

If you will be doing database maintenance, you can close the database and leave the server running. This way people will get a friendly message when they attempt to access the database. Just create the file */etc/smsdown* on the server, and the next time the server has no active connections it will close the database. Thereafter, if anyone starts a moira client, the contents of this file will be displayed as the moira ‘‘message of the day’’. If you later remove the file, the next time a client connects to the server it will reopen the database. If you are in a hurry to close or open the database, you can send it a signal USR1 (i.e. `kill -USR1` *pid*) to make it close as soon as the current query is finished processing, or a signal USR2 (i.e. `kill -USR2` *pid*) to make it open immediately. Note that a side effect of closing and reopening the database is to flush all cached information such as query access control lists or name to

ID mappings.

To actually kill the server, simply send it a HUP or TERM signal (i.e. `kill` *pid*). It will then exit as soon as the current query finishes processing. If you want to shutdown more gracefully, either wait until no clients are connected to the server (by monitoring */moira/moira.log* or using the `_list_users` query) and then kill the server or first close the database as specified above, then kill the server once the database has closed. Note that the server may not exit right away since it will attempt to finish up any query currently being processed and drain the incremental update work queue as well.

5.2. Accounts

There are a number of things the system manager must do to keep user account management running smoothly. These include loading new users into the database, deleting old accounts, and some general cleanup.

5.2.1. User Account Tapes

Much of the general cleanup and maintenance of user accounts is done via the **students** and **employee** programs. These programs each process tapes of information obtained from the registrar and personnel offices. Operation of each program is the same, although the format of the two tapes is slightly different.

The program processes records one at a time. It takes the input records and computes the encrypted ID number, full name in mixed case, and class. It then uses the encrypted ID as a key to lookup the user. If the user is not already in the database, he is added. If he is already there, the class, status, and address fields are checked and updated if necessary.

The status field will be set depending on the class. Students and faculty are eligible for accounts and are created in state 0 (registerable). Others are created in state 4 (not enrolled).

Whenever a user is touched, either because they were found on the tape or because they are added as a new user, these programs set the *ugdefault* field to 1. This way if these fields are all cleared before loading tapes, afterwards you have a record of who was on the latest tape. It is generally safe to delete users in states 0 or 4 (unregistered or unenrolled) who are not on the latest tapes. And since these users never touched their entries, there are no other references to them in the database to clean up.

There are several options to these programs that are generally used: **-n** causes moira to fill in the finger information for users who currently have those fields blank. **-u** causes an additional relation to be maintained in the database (called ‘‘xusers’’) which contains all of the information from the tape, including the unencrypted ID number.

5.2.2. Register

There are three components used in registration: the **reg_svr** daemon running on the moira server machine, the **userreg** client program, and a shell script necessary to get **userreg** started correctly under different circumstances. [Athena] For instance, one can login for a tty-based session as user ‘‘register’’, password ‘‘athena’’ and the user will be prompted for a terminal type, then dropped into **userreg**. Or they may press the ‘‘Click here to register for an account’’ button on our current login screen, which runs a script to start **userreg** in an xterm. In a previous configuration clicking on that button logged the user in as register for a workstation session instead of a tty-based session, requiring another script. These will have to be customized for your workstation configuration.

When **userreg** is started, it checks a file called *disabled.times* to determine if anyone is allowed to register right now. This file contains *crontab*-style lines specifying times when registration is not allowed and the appropriate message to display at those times. For example, the file:

```
* 23 * * *       nightly database propagation is in progress.
0-29 0 * * *     nightly database propagation is in progress.
* 4 * * 7        weekly database maintenance is in progress.
```

Would disallow registration from 11:00pm to 12:29am each evening, and from 4:00am to 4:59am on Sunday mornings. Do not put in a line with all asterisks to match all times, as this will confuse the algorithm **userreg** uses to tell the user when the next time is that registration will be allowed. It is a good idea to disable registration this way whenever taking down the **reg_svr** since otherwise users will not get an informative message as to why they cannot register (they will simply be told that ‘‘part of the computer system is down’’).

The **userreg** program enforces the same capitalization rules as the bulk load programs (described on page 6). Even though the name may occasionally not look right on the screen, that should be the way it is recorded on the database. The first name, last name, and ID number must all match what is in the database before the user will be found.

The suggested username is a concatenation of the first initial, middle initial, and first six characters of the last name. However, users are free to choose any username that begins with a lowercase letter, is followed by lowercase letters, numerals, or underscore, and is from three to eight characters long. **Userreg** first checks to see if the proposed username is in use in kerberos before querying moira (since the kerberos lookup is much faster). If you have kerberos principals without corresponding accounts in moira, those usernames will still be reserved in moira.

Setting the username in the database is the heart of the registration process, as all other resources for the account are set at the same time. The **reg_svr** simply executes a *register_user* query and reserves the principal in kerberos. This query takes the following actions:

1. The user must exist and have a status of 0 (not registered), or 6 (enrolled but not registered).
2. The new login name must not be in use as a login name, list name, or filesystem name.
3. A location is chosen for the user’s pobox, and this pobox is assigned.
4. A ‘‘group’’ list is created with the same name as the user, which is an active, private, visible group with a unique GID. The administrator is the user and it has one member who is also the user.
5. A partition is chosen for the user’s home directory, and this locker is then created with the same name as the user, owned by the user, with locker type HOMEDIR.
6. The default quota is assigned for the user on the locker.
7. The user is left in state 2 (half-registered).

Only if all of these steps are successful will the new username be allowed.

The pobox is placed on the POP server with the most free space. This is found by scanning the server-host tuples for each of the hosts supporting service POP. *Value1* for a POP server indicates how many poboxes are assigned to this server, and is maintained automatically by moira. *Value2* is the maximum number of poboxes allowed on this server. The POP server is chosen which has the largest difference between the maximum and the current usage.

The home directory is located in a similar manner. The NFS partitions are scanned. The use bits of each one are compared with the usertype argument to *register_user* (the *reg_svr* always uses type 1 (student)). For each one supporting the correct kind of user, the difference between the size and the allocated quota is determined. The eligible partition with the largest difference will be used. The quota assigned is the value of ‘‘def_quota’’ stored in the moira values table.

When this step is completed, the user is half-registered. They still have to choose a password. When they enter their password, *reg_svr* will set it in kerberos and update their status to 1 (active). Their account is now complete, and will be usable the next time moira updates the system services.

5.2.3. General Cleanup

It is good to periodically check the database to see that things are in order. Things to check for include: users without poboxes, duplicates, users without filesystems, obscure classes, etc. Most of this is done directly in SQL. Some things, like deleting users who never registered for their accounts and are no longer eligible, are easy and should be done monthly when new tapes are loaded. Other cleanup only needs to be done once a term or once a year.

To delete a mass of users, simply use the interactive SQL commands to delete the desired columns from the *users* table. Then run **dbck** to catch and remove any dangling references. You will probably want to delete any filesystems and lists with the same name as the deleted users.

[Athena] The procedure for deactivating accounts here at Athena is as follows:

1. For the first month of the academic year, we make a list of everyone who used the system by processing the kerberos logs.
2. We load the latest tapes from the registrar and personnel offices, flagging which accounts were not on those tapes.
3. Decide which values in the class field of the user record are targets for removal. Generally this is the undergrads who should have just graduated, grad students whom the registrar says aren't students anymore, and special accounts from the previous year. All others are examined by hand.
4. Accounts which are targeted for deletion and weren't used in the last month have their status set to 3. This causes the account to no longer be usable, but it can easily be re-enabled by changing the status back to 1.
5. Accounts which are targeted for deletion but are still being used have mail sent to them informing them that their accounts will be turned off in two weeks. During this time they can appeal this decision to the accounts consultants.
6. After the two weeks, anyone who has not had their class changed to GUEST or otherwise been saved from the purge will have their status set to 3.
7. After a couple of months, we migrate the home directories of all of the accounts in state 3 to one partition, make an archival copy of the partition, then delete their filesystems. It is no longer simple to reactivate their accounts.
8. The following summer, we really delete them. By waiting most of a year, we reserve the username and UID for that period of time to avoid confusion in email or file ownership. At deletion time, we use raw SQL to delete any user record in state 3 that has been unmodified for more than 6 months, and delete the corresponding lists. The moira database consistency checker is run to catch any dangling references to these now-deleted accounts.

5.3. Updates

Keeping tabs on the regular moira updates of system servers is one of the daily duties of the moira system administrator. Normally, everything works without any intervention. However, when errors do occur, the system will not attempt any more updates for the failed service until the error is manually reset.

Updates are attempted whenever the data control manager, or **dcm**, is run. The **dcm** is normally invoked by **cron**. You may configure it to run regularly, perhaps as often as every 15 minutes, or only run **dcm**s at certain times of the day. [Athena] Because the database is read-locked for the duration of a service update, we only run updates at night.

This means that changes made during the day are not visible until the following day. We invoke the **dcm** hourly between 11pm and 5am. Invoking it many times gives parallelism for updates that take a long time and guarantees retries in the case of transient failures.

Cron actually runs **startdcm** , which is a program that launches the dcm after connecting to the root directory and setting up the debugging and logging correctly. It is **startdcm** which puts the timestamp on each line logged to */moira/dcm.log*. It is not necessary to use **startdcm**; it is a convenience program to avoid having to always have the proper environment to start the **dcm**.

5.3.1. Services to be updated

For each service moira supports, the following information is kept:

name	Up to 16 characters long, case in-sensitive.
update interval	The minimum amount of time between updates of this service. Displayed as hh:mm:ss, but new values are input in minutes only.
target file	Where on each server to put the file generated.
script	Name of the script file on the moira server to be sent over to the server and executed during the server update. [Athena] This is usually /moira/bin/*service*.sh.
When last generated	This read-only field shows when data files for this service were last generated.
When last checked	This read-only field shows when the **dcm** last checked this service to see if it needed updating.
type	Service type, used to determine how to deal with concurrency and errors during the update. [Athena] Types are UNIQUE and REPLICATED.
enable/disable	This flag indicates if the **dcm** should attempt updates of this service.
idle/inprogress	This read-only flag indicates if the **dcm** is currently generating data files for this service.
hard error	no error This flag indicates that this service had an error the last time an update was attempted. It is set by the **dcm**, and may only be cleared by the user.
error message	This field will contain the error if the *hard error* flag is set.
access control	This field consists of a type and a name. If the type is USER, the name must be a login name. If the type is LIST, the name must name a list. The type may be NONE, in which case the name is ignored. A user on the access control list may reset service errors and manipulate hosts supporting the service.

When the **dcm** runs, it scans the services, finding those which are enabled and do not have hard errors, and more than update interval minutes have passed since the time data files were last generated for this service. For each of these services, it will set the time last checked to now, then run the generator for that service. The generator for service *foo* is /moira/bin/*foo*.gen. The generator may be successful, in which case it leaves behind /moira/dcm/*foo*.out and sets the time last generated. It may decide that the database is unchanged and not regenerate the data files. It may get a soft error, meaning that if the **dcm** tries again it may succeed. It will record the soft error in the error message field, but not set the hard error flag. It may get a hard error, in which case it sets the hard error flag and records the error message. While the **dcm** is working on a service, it obtains an exclusive lock on that service so that another **dcm** running will not also attempt to work on the same service.

Errors that occur during service updates are often caused by the machine the moira server is running on running out of some resource, such as swap space or maximum number of database locks. These are not always caught and marked as soft errors because they often cause the generator program to exit with an unexpected status. Also note that the error message recorded with a failure may not be correct as only the 8 least significant bits are passed from the generator to the **dcm**, which must guess as to what range that code was supposed to fall into. If the cause of an error is not immediately apparent, check the log file /moira/dcm.log.

5.3.2. Hosts to be updated

For each host that supports a given service, moira contains the following information:

service	Name of service this host is supporting. This field is case insensitive.
host	Name of machine, which must already be known to moira.
success/failure	This read-only flag indicates if the last attempted update was successful or not.
enable/disable	This flag indicates if updates should be attempted for this host.
override/normal	This indicates a priority on the next update. Setting the override flag causes a **dcm** to be started immediately, and this service/host will updated even if the service interval has not yet passed. This flag may be set by the user, but is only cleared by the **dcm**.
inprogress/idle	This read-only flag indicates that an update is in progress to this host.
hard error/no error	This flag indicates that this host had an error the last time an update was attempted. It is set by the **dcm**, and may only be cleared by the user.
error message	This field will contain the error if the *hard error* flag is set.
last time tried	This read-only field shows when the last update was attempted.
last time successful	This read-only field shows when the last successful update was.
value1	This integer field has a service-dependent value.
value2	This integer field has a service-dependent value.
value3	This 32 character string field has a service-dependent value.

As the **dcm** scans the services, it will also scan the hosts supporting each service that is enabled. Before starting the host scan, if the service is of type REPLICATED it gets a an exclusive lock on that service and if it is of type UNIQUE it gets a shared lock. After obtaining locks, it will check each host supporting that service which is enabled, doesn't have a hard error, and hasn't been updated since the last time the data files were generated for this service. If the *override* flag is set, it will attempt the host even if it has been updated since the data files were generated. The **dcm** gets an exclusive lock on that host, then attempts the update.

The update consists of sending over the file /moira/dcm/*service*.out from the moira server machine to the *target* filename on the host, then sending over the *script* to a temporary file and then executing the script. If a soft error occurs, the error message will be recorded and everything else left so that the next **dcm** pass will try again. If a hard error occurs, the hard error flag will be set as well. If the service is of type REPLICATED, the service will be marked with a hard error as well so that no further updates will be attempted of the replicated service.

Common errors to be on the watch for include network failures, the hosts being down, and the hosts running out of disk space to accept the update. The error message may explain what went wrong, but when in doubt check /moira/dcm.log on the moira server machine.

In addition to controlling updates, hosts supporting services are also loaded into the nameserver with type *sloc*. Thus querying hesiod for (nfs, sloc) will return a list of the machines moira sends NFS updates to. If you want to put an *sloc* entry into the nameserver without actually doing updates for that service, make an entry for the service which is disabled, then entries for each of the hosts which are enabled. Most of the information will be ignored during updates, and the nameservice will get the correct info.

5.4. Database Maintenance

Backups are a part of regular database maintenance. If your system changes regularly, you should be doing daily backups of the moira database. You may do UNIX backups of the database partition, DBMS backups of the database, or use the programs provided with moira. The advantage to using the moira programs **mrbackup** and **mrrestore** is that they save the database in an ASCII format which is easy to work with and independent of the DBMS. They also guarantee consistency of the dump through the use of database transactions.

Mrbackup is simple to run, taking only one argument indicating a directory to put the backup in. It will produce a file for each table in the database. These files contain a line for each record, the lines consisting of vertical bar-separated fields. [Athena] At Athena, we run **mrbackup** to another partition on the moira server, and copy that backup over the net to two other machines. The backups are not regularly copied to tape, partly because the privacy-sensitive information in the database would require keeping closer control on those tapes than the regular user file backups.

In the event you ever need to restore a database, here are some instructions on how to do that.

1. SU to whatever user should own the database.
2. Create an empty database named *smstemp*.
3. Run the *db/newdb* script to create all of the tables in the database.
4. Run **mrrestore**, giving it an argument of the directory name containing the backup.
5. Run the *db/dbopt* script to create all of the indexes on the tables.
6. Run the appropriate DBMS commands to set permissions on the tables. [Athena] For Ingres, this would be ‘‘define permit all on *table* to root’’ for each table.
7. Skim the journal file and enter any changes needed between the time of the last backup and when the database was lost.

5.4.1. Granting Privileges

There are a number of ways that privileges can be granted in moira. At the lowest level is the privilege of performing QUEL or SQL operations directly on the database. To grant this privilege in Ingres, go into **quel** and enter the command

`define permit all on` *table* `to` *user*

for each of the tables you want to allow *user* to have access. Note that **newmoira** will do this for you at the time you originally create the moira database.

More commonly, you will be concerned with the privileges necessary to perform moira queries through the clients. Each of the 130 queries has an access control list associated with it. These lists are references to regular moira lists which may be manipulated with the regular moira list tools (e.g. **listmaint**, **blanche**, etc). If a user is on the ACL for a query, he can perform that query on any data in the database. Many queries also have conditional ACLs based on the object of the query. For example, being on the ACL for get_members_of_list will allow you to get the membership of any list, whereas without being on that ACL you can still get the membership of any list of which you are an administrator, or any list which is not hidden.

Some thought needs to be put into how you assign the query ACLs. The **newmoira** program sets all of them to be the list *dbadmin* which it sets up to contain each of the privileged users you name. This gives the users on this list all privileges, and denies any privilege to other users. Some queries are safe to be executed by anyone, and there is a way to specify that: any list containing the user ‘‘default’’ is considered to have everyone as a member for the purposes of checking query ACLs. [Athena] They queries we have set up this way are:

- get_alias
- get_list_is_group
- get_list_is_maillist
- get_server_host_info
- get_server_info
- get_server_locations
- get_value
- qualified_get_server_host
- qualified_get_server

Many queries you may want to set up with a larger ACL than just the database administrators, but without everybody. Because these are a little difficult to change, have a rational plan in mind for setting up ACLs rather than just assigning them as they are needed. If you want to continue having the people on *dbadmin* have full privileges, then you will have to make sure that LIST dbadmin is a member of each ACL that you change.

To set a query ACL:

1. Create a list to be the ACL, or choose an existing list. For example, ‘‘user-accts’’.
2. Go into **quel** on the moira server
   ```
   quel sms
   ```
3. Find the *list_id* of the list:
   ```
   range of l is list
   retrieve (l.list_id) where l.name="user-accts"
   ```
4. Update the ACL:
   ```
   range of c is capacls
   replace c (list_id=1837) where
           c.capability="get_user_by_login"
   ```
5. Kill and restart the **moirad** so the change will actually take effect

5.4.2. ID Number Allocation

Moira handles a number of different kinds of ID numbers, both for its own use internally and externally visible ones like UIDs and GIDs. There are entries in the *values* table for each ID number indicating the next number to be allocated. When moira needs another number, it reads the current value out of the values table, and checks to see if that ID is in use. If not, it increments the value stored in the values table and uses the old value. If that value was already in use, it increments the value and checks the next one. Each time it increments the value, it compares it with the maximum ID, 32765. If it has reached this, it sets the value to 100 and continues from there.

This is usually fast to assign ID numbers. However, if the assignment value has wrapped and it must do many unsuccessful checks before it finds a number to use, this may take a while. You may set the value in the values table if you want to change the range that ID numbers are being assigned in.

5.4.3. Type Allocation

A number of the fields in the database are type-checked against a list of legal values also stored in the database. These are stored in the alias table as aliases of type TYPE. The **moira** client will automatically assign new values if the user insists that he really wants to add a new legal value. The database administrator may occasionally want to clean up the values and delete ones no longer in use. This is done by scanning the output of the query

```
get_alias * TYPE *
```

and deleting any aliases that shouldn’t be there.

Index

Table of Contents