Moira Queries

Draft of 21 May 1990

The following descriptions contain the long name of the query, the short name, any arguments in parenthesis, then a "=>" followed by any return values. Zero or more sets of values may be returned. The three values "modtime, modby, modwith" return the date and time of the last modification, the name of the kerberos principal who made the modification, and the name of the program that performed the query.

If a description says nothing about access control, then it may only be executed by people on the ACL for that query. Most query ACLs contain a small number of people, the "gods" for that query. There is also a magic list called "default" (the fact that it contains the user named "default" is actually the magic) which is treated as if it contained the name of every user. This list is used as the query ACL for some public queries.

Any query may return: MR_PERM "Insufficient permission to perform requested database access" or MR_ARGS "Incorrect number of arguments". Any retrieval query may return MR_NO_MATCH "No records in database match query". Any add or update query may return MR_BAD_CHAR "Illegal character in argument" if a bad character is in an argument that has character restrictions, or MR_EXISTS if the new object to be added or new name of existing object conflicts with another object already in the database. Other errors are listed with each query.

Users, Finger, and Post Office Boxes

get_all_logins, galo() => *login, uid, shell, last, first, mi*

Returns info on every account in the database. The returned info is a summary of the account info, not the complete information.

get_all_active_logins, gaal() => *login, uid, shell, last, first, mi*

Returns info on every account for which the status field is non-zero. The returned info is a summary of the account info, not the complete information.

get_user_by_login, gubl(*login***) =>** *login, uid, shell, last, first, mi, state, mitid, class, modtime, modby, modwith*

Returns complete account information on the named account. Wildcards may be used in the *login* name specified. If the person executing the query is not on the query ACL, then the query only succeeds if the only retrieved information is about the user making the request.

get_user_by_uid, gubu(*uid***) =>** *login, uid, shell, last, first, mi, state, mitid, class, modtime, modby, modwith*

Returns complete account information on any account with the specified *uid*. If the person executing the query is not on the query ACL, then the query only succeeds if the only retrieved information is about the user making the request.

get_user_by_name, gubn(*first, last***) =>** *login, uid, shell, last, first, mi, state, mitid, class, modtime, modby, modwith*

Returns complete account information on any account with matching *first* and *last* name fields. Either or both names may contain wildcards, so that this query can do the equivalent of lookup by firstname or lookup by lastname.

get_user_by_class, gubc(*class***) =>** *login, uid, shell, last, first, mi, state, mitid, class, modtime, modby, modwith*

Returns complete account information on any account with a matching class field. The given class may contain wildcards.

get_user_by_mitid, gubm(*crypt(id***)) =>** *login, uid, shell, last, first, mi, state, mitid, class, modtime, modby, modwith*

Returns complete account information on any account with a matching MIT ID field. The given id may contain wildcards.

add_user, ausr(*login, uid, shell, last, first, mi, state, mitid, class***) =>**

Adds a new user to the database. *login* must not match any existing *login*s. *uid* and *state* must be integers. If the given uid is **UNIQUE_UID** as defined in <*moira.h*>, the next unused uid will be assigned. If *login* is **UNIQUE_LOGIN** as defined in <*moira.h*>, the login name will be a "#" followed by the uid. For example, when adding a person so that they may register later, the query ausr(UNIQUE_LOGIN, UNIQUE_UID, /bin/csh, *Last, First, M*, 0, [encrypted ID], *class*) is used. The *class* field must contain a value specified as a **TYPE** alias for **class**. This query also initializes the finger record for this user with just their full name, and sets their pobox to **NONE**. It updates the modtime on the user, finger and pobox records. Errors: MR_NOT_UNIQUE "Arguments not unique" if the login name is not unique, or MR_BAD_CLASS "Specified class is not known" if the class is not in the alias database.

register_user, rusr(*uid, login, fstype***) =>**

Registers a user. This consists of changing their username, and creating a pobox, a group list, a filesystem, and a quota for them. The user is identified by *uid*, which must match exactly one existing user. Further, this user must currently have a *status* of 0. The user will be left with a *status* of 2. The pobox created will be of type **POP** on the least loaded post office. The group list will have the user as an owner, and a unique GID will be assigned. The filesystem will be allocated on the least loaded fileserver which supports *fstype*, where *fstype* is **MR_FS_STUDENT, MR_FS_FACULTY, MR_FS_STAFF**, or **MR_FS_MISC** as defined in <*moira.h*>. A quota will be assigned to the user on his filesystem with the value taken from **def_quota** in the values table. Errors: MR_NO_MATCH, MR_NOT_UNIQUE "Arguments not unique" if the uid does not specify exactly one user; MR_IN_USE "Object is in use" if the login name is already taken.

update_user, uusr(*login, newlogin, uid, shell, last, first, mi, state, mitid, class***) =>**

Updates the info in a user entry. *login* specifies the existing login name, the remaining arguments will replace the current values of those fields. This is not equivalent to deleting the user and adding a new one, as all references to this user will still exist, even if the login name is changed. All fields must be specified, even if the value is to remain unchanged. *login* must match exactly one user in the database. *newlogin* must either match the existing login or be unique in the database. The *class* field must contain a value specified as a **TYPE** alias for **class**. *uid* and *state* must be integers. The modtime fields in the user's record will be updated. Errors: MR_USER "No such user" if the login name does not match exactly one user, MR_NOT_UNIQUE "Arguments not unique" if the new login name is not unique, or MR_BAD_CLASS "Specified class is not known" if the class is not in the alias database.

update_user_shell, uush(*login, shell***) =>**

Updates a user's shell. *login* must match exactly one user. The modtime fields in the user's record will be updated. This query may be executed by the target user or by someone on the query ACL. Errors: MR_USER "No such user" if the login name does not match exactly one user.

update_user_status, uust(*login, status***) =>**

Updates a user's *status*. *login* must match exactly one user. The modtime fields in the user's record will be updated. Errors: MR_USER "No such user" if the login name does not match exactly one user.

delete_user, dusr(*login***) =>**

Deletes a user record. *login* must match exactly one user. **The user must have a status of zero, or MR_IN_USE will be returned.** This will only be allowed if the user is not a member of any lists, has any quotas assigned, or is the owner of an object. It will also delete associated finger information, post office box, and any quotas the user has. Errors: MR_USER "No such user" if the login name does not match exactly one user, MR_IN_USE "Object is in use" if the user is a member of a list, has a quota or is an ACE.

delete_user_by_uid, dubu(*uid***) =>**

Deletes a user record. *uid* must match exactly one user. This will only be allowed if the user is not a member of any lists or is the owner of an object. It will also delete associated finger information and post office box. Errors: MR_USER "No such user" if the uid does not match exactly one user, MR_IN_USE "Object is in use" if the user is a member of a list or is an ACE.

get_kerberos_user_map, gkum(*login, kerberos***) =>** *login, kerberos*

Gets kerberos principal to user mapping entries. Both *login* and *kerberos* may contain wildcards. A user may only retrieve his own entry unless he is on the query ACL.

add_kerberos_user_map, akum(*login, kerberos***) =>**

Adds a new kerberos principal to user mapping entry. *Login* must match exactly one user. *Kerberos* is a case-sensitive name/instance/realm principal string. These mappings must be strictly one-to-one, so a new entry will not be allowed if the user or principal already appears in the mapping table. A user may only add a mapping for himself unless he is on the query ACL. Errors: MR_USER "No such user" if *login* does not match exactly one user, MR_EXISTS "Record already exists" if the user or principal already has a mapping.

delete_kerberos_user_map, dkum(*login, kerberos***) =>**

Deletes a kerberos principal to user mapping. *Login* must match exactly one user. *Kerberos* must exactly match an existing mapping for that user. A user may only remove his own mapping unless he is on the query ACL. Errors: MR_USER "No such user" if *login* does not match exactly one user.

get_finger_by_login, gfbl(*login***) =>** *login, fullname, nicname, home_addr, home_phone, office_addr, office_phone, department, affiliation, modtime, modby, modwith*

Gets all of the finger information for the specified user. *login* must match exactly one user. The target user may retrieve his information. It is safe to point the query ACL at the list of all users. Errors: MR_USER "No such user" if the login name does not match exactly one user.

update_finger_by_login, ufbl(*login, fullname, nicname, home_addr, home_phone, office_addr, office_phone, department, affiliation***) =>**

Allows any part of the finger information to be changed for a specified account. *login* must match exactly one user. The remaining fields are free-form, and may contain anything. The modtime fields in the finger record will be updated. A user may update his own information. Errors: MR_USER "No such user" if the login name does not match exactly one user.

get_pobox, gpob(*login***) =>** *login, type, box, modtime, modby, modwith*

Retrieves a user's post office box assignment. The *login* name must match exactly one user. See *set_pobox* for a summary of the returned fields. The owner of the pobox may perform this query. Errors: MR_USER "No such user" if the login name does not match exactly one user.

get_all_poboxes, gapo() => *login, type, box*

Retrieves all of the post office boxes from the database. See *set_pobox* for a summary of the returned fields.

get_poboxes_pop, gpop() => *login, type, machine*

Retrieves all of the post office boxes of type **POP** from the database. See *set_pobox* for a summary of the returned fields.

get_poboxes_smtp, gpos() => *login, type, box*

Retrieves all of the post office boxes of type **SMTP** from the database. See *set_pobox* for a summary of the returned fields.

set_pobox, spob(*login, type, box***) =>**
Establishes a user's post office box. The given *login* must match exactly one user. The *type* will be checked against the alias database for valid **pobox** types. Currently allowed types are **POP**, **SMTP**, and **NONE**. If the type is **POP**, then box must name a machine known by MR. If the type is **SMTP**, then box is the user's mail address with no other interpretation by MR. A type of **NONE** is the same as not having a pobox. The modtime fields on the pobox record will be set. The owner of the target pobox may perform this query. Errors: MR_USER "No such user" if the login name does not match exactly one user, or MR_TYPE "Invalid type" if the type is not **POP**, **SMTP**, or **NONE**.

set_pobox_pop, spop(*login***) =>**
Forces a user's pobox to be type **POP**. The *login* name must match exactly one user. If the user's pobox is already of type **POP**, nothing will be changed. If the user has previously had a pobox of type **POP**, then the previous post office machine assignment will be restored. If there was no previous post office assignment, the query will fail with MR_MACHINE "Unknown machine" since it will be unable to choose a post office machine. The modtime fields on the pobox record will be set. The owner of the target pobox may perform this query. Errors: MR_USER "No such user" if login does not match exactly one user, or MR_MACHINE.

delete_pobox, dpob(*login***) =>**
Effectively deletes a user's pobox, by setting the type to **NONE**. The *login* name must match exactly one user. The modtime fields on the pobox record will be set. The owner of the target pobox may perform this query. Errors: MR_USER "No such user" if login does not match exactly one user.

Machines and Clusters

get_machine, gmac(*name***) =>** *name, type, modtime, modby, modwith*
Get all the information on the specified machine(s). Wildcarding may be used in the machine *name*. All machine names are case insensitive, and are returned in uppercase. It is safe for the query ACL to be the list containing everybody.

add_machine, amac(*name, type***) =>**
Enters a new machine into the database. The given *name* will be converted to uppercase. Then it will be checked for uniqueness in the database. The *type* field will be checked against the aliases database for valid **mach_type**s. Currently defined **mach_types** are **RT** and **VAX**. The modtime fields will be set. Errors: MR_NOT_UNIQUE "Arguments not unique" if a machine with the given *name* already exists, or MR_TYPE "Invalid type" if the given *type* is not in the alias database.

update_machine, umac(*name, newname, type***) =>**
Update the information on a machine. The *name* must match exactly one machine. The *newname* must either be the same as the old name, or must be unique among machine names in the database after being converted to uppercase. The type field will be checked against the aliases database for valid **mach_type**s. The modtime fields will be set. Errors: MR_MACH "No such machine" if the old *name* does not match exactly one machine, MR_NOT_UNIQUE "Arguments not unique" if the *newname* does not either match the old name or is unique, or MR_TYPE "Invalid type" if the given *type* is not in the alias database.

delete_machine, dmac(*name***) =>**
Delete a machine from the database. The given *name* must match exactly one machine. A machine that is in use (post office, file system, printer spooling host, server_host_access, or DCM service update) cannot be deleted. Errors: MR_MACH "No such machine" if the *name* does not match exactly one machine, or MR_IN_USE "Object is in use" if the machine is being referenced as a post office, filesystem, spooling host, or server updated by the DCM.

get_cluster, gclu(*name***) =>** *name, description, location, modtime, modby, modwith*

Returns all the information in the database about one or more clusters. The cluster *name* may contain wildcards. It is safe for the query ACL to be the list containing everybody.

add_cluster, aclu(*name, description, location***) =>**

Adds a new cluster to the database. The *name* must be unique among the existing cluster names. The names are case sensitive. There are no constraints on the remaining data. The modtime fields will be set. Errors: MR_NOT_UNIQUE "Arguments not unique" if the cluster *name* is not unique.

update_cluster, uclu(*name, newname, description, location***) =>**

Changes the information about a cluster. The old *name* must match exactly one cluster. The *newname* must either match the old name or be unique among the existing cluster names. The names are case sensitive. There are no constraints on the remaining data. The modtime fields will be set. Errors: MR_CLUSTER "Unknown cluster" if the old cluster name does not match exactly one cluster, or MR_NOT_UNIQUE "Arguments not unique" if the new name does not either match the old name or is unique.

delete_cluster, dclu(*name***) =>**

Removes a cluster from the database. The *name* must match exactly one cluster. The cluster must not have any machines assigned to it. Any service cluster information assigned to the cluster will be deleted. Errors: MR_CLUSTER "Unknown cluster" if the old cluster name does not match exactly one cluster, or MR_IN_USE "Object in use" if the cluster has machines assigned to it.

get_machine_to_cluster_map, gmcm(*machine, cluster***) =>** *machine, cluster*

Retrieves machine to cluster mappings for the specified *machine(s)* and *cluster(s)*. Either of the fields may contain wildcards. It is safe for the query ACL to be the list containing everybody.

add_machine_to_cluster, amtc(*machine, cluster***) =>**

Add a machine to a cluster. The *machine* name must match exactly one machine. The *cluster* name must match exactly one cluster. The machine's modtime fields will be updated. Errors: MR_MACHINE "No such machine" or MR_CLUSTER "No such cluster" if one of them does not match exactly one object in the database.

delete_machine_from_cluster, dmfc(*machine, cluster***) =>**

Delete a machine from a cluster. The *machine* name must match exactly one machine. The *cluster* name must match exactly one cluster. The named machine must belong to the named cluster. The machine's modtime fields will be updated. Errors: MR_MACHINE "No such machine" or MR_CLUSTER "No such cluster" if one of them does not match exactly one object in the database.

get_cluster_data, gcld(*cluster, label***) =>** *cluster, label, data*

Retrieve all cluster data matching the named *cluster* and *label*. Either or both may use wildcards. Thus all data for a cluster may be retrieved with gcld(cluster, *), and all data of a particular type may be retrieved with gcld(*, label). It is safe for the query ACL to be the list containing everybody.

add_cluster_data, acld(*cluster, label, data***) =>**

Add new data to a cluster. The *cluster* name must match exactly one cluster. The service *label* must be a registered **slabel** in the alias database. The *data* is an arbitrary string. The cluster's modtime fields will be updated. Errors: MR_CLUSTER "No such cluster" if the *cluster* name does not match exactly one cluster, or MR_TYPE "Invalid type" if the *label* is not in the alias database.

delete_cluster_data, dcld(*cluster, label, data***) =>**

Delete the specified cluster data. The *cluster* name must match exactly one cluster, and the remaining two arguments must exactly match an existing service cluster. The cluster's modtime fields will be updated. Errors: MR_CLUSTER "No such cluster" if the *cluster* name does not match exactly one cluster, or MR_NOT_UNIQUE "Arguments not unique" if the *label* and *data* do not match exactly one existing piece of data for the cluster.

Lists

get_list_info, glin(*list***) =>** *list, active, public, hidden, maillist, group, gid, ace_type, ace_name, description, modtime, modby, modwith*

Returns information about the named list. The *list* name may contain wildcards. *active, public, hidden, maillist*, and *group* are booleans returned as integers (0 is false, non-zero is true). The *ace-type* is either **USER**, **LIST**, or **NONE**, and the *ace_name* will be either a login name, a list name, or **NONE**, respectively. This query is allowed if the list is not hidden or the user executing the query is on the ACE of the target list. If the user executing this query is on the query ACL, he may use wildcards in the *list* name, otherwise wildcards are not allowed.

expand_list_names, exln(*list***) =>** *list*

Expands wildcards in a *list* name. A name is passed which may contain wildcards, and a set of matching names are returned.

add_list, alis(*list, active, public, hidden, maillist, group, gid, ace_type, ace_name, description***) =>**

Creates a new list and adds it to the database. The *list* name must be unique among existing list names. *active, public, hidden, maillist,* and *group* are booleans passed as integers (0 is false, non-zero is true). If *group* is true and *gid* is **UNIQUE_GID** as defined in <*moira.h*>, a new unique group ID will be assigned, otherwise the integer value given for *gid* will be assigned to the GID. The *ace-type* is either **USER**, **LIST**, or **NONE**, and the *ace_name* will be either a login name, a list name, or **NONE**, respectively. The access list may be the list that is being created (self-referential). The list modtime will be set. Errors: MR_EXISTS "Record already exists" if a list already exists by that name, MR_ACE "No such access control entity" if the *ace_type* is not **USER**, **LIST**, or **NONE**, or if the *ace_name* cannot be resolved relative to the *ace_type*.

update_list, ulis(*list, newname, active, public, hidden, maillist, group, gid, ace_type, ace_name, description***) =>**

Allows the list information and attributes to be changed. This is not equivalent to deleting the list and creating a new one, since references to the old name will still apply to the new name if it is renamed. The *list* name must match exactly one list. The *new name* must either match the old name or be unique among list names in the database. The *active, public, hidden, maillist*, and *group* flags should be 0 if the flag is false, or non-zero if it is true. The *gid* may be **UNIQUE_GID** as defined in <*moira.h*>, in which case a new unique GID will be assigned. The *ace-type* is either **USER**, **LIST**, or **NONE**, and the *ace_name* will be either a login name, a list name, or **NONE**, respectively. The list modtime will be set. This query may be executed by anyone on the ACE of the target list. Errors: MR_LIST "No such list" if the named list does not match exactly one list, MR_NOT_UNIQUE "Arguments not unique" if the new list name doesn't either match the old one or is unique, MR_ACE "No such access control entity" if the *ace_type* is not **USER**, **LIST**, or **NONE**, or if the *ace_name* cannot be resolved relative to the *ace_type*.

delete_list, dlis(*list***) =>**

Deletes a list from the database. A list may only be deleted if it is not in use as a member of any other list or as an ACL for an object, and the list itself must be empty. This query may be executed by someone who is on the current ace of the target list. Errors: MR_LIST "No such list" if the named list does not exist, or MR_IN_USE "Object is in use" if the list is still being referenced.

add_member_to_list, amtl(*list, type, member***) =>**

Adds a member to a list. The specified *list* must match exactly one list. Type must be either **USER**, **LIST**, or **STRING**. *member* is either a login name, a list name, or a text string, respectively. The modtime on the list is updated. This query may be executed by: anyone adding themselves as a *USER* to a list with the *public* bit set or anyone on the ACE of the list being modified. Errors: MR_LIST "No such list" if the named list does not exist, MR_TYPE "Invalid type" if the member *type* is not **USER**, **LIST**, or **STRING**, or MR_NO_MATCH "No records in database match query" if the *member* name cannot be resolved with the member *type*.

delete_member_from_list, dmfl(*list, type, member***) =>**

Deletes a member from a list. The specified *list* must match exactly one list. The specified *type* and *member* must exactly match an existing member of that list. The modtime on the list is updated. This query may be executed by anyone deleting themselves as a **USER** from a list with the *public* bit set or anyone on the ACE of the list being modified. Errors: MR_LIST "No such list" if the named list does not exist, MR_TYPE "Invalid type" if the member type is not **USER**, **LIST**, or **STRING**, or MR_NO_MATCH "No records in database match query" if the *member* name cannot be resolved with the member *type* or if there is no such member in the list.

get_ace_use, gaus(*ace_type, ace_name***) =>** *object_type, object_name*

Finds references to an object as an ACE. Valid *ace_types* are **USER**, **LIST**, **RUSER**, and **RLIST**. For types **USER** and **RUSER**, the *ace_name* must be a login name. If the type is **USER**, then only objects whose ACE is the named user will be found; if it is **RUSER**, it will recursively check down sub-lists of the ACE lists looking to see if the user is a member of that ACE. The types **LIST** and **RLIST** apply to a list name in a similar manner. The returned tuples will be **LIST**, list name; **SERVICE**, service name; **FILESYS** and a filesystem label; **QUERY**, query name; **HOSTACCESS**, machine name; or **ZEPHYR**, zephyr class name. This query may be executed by a user asking about himself or a person on an ACE of a list asking about that list. Errors: MR_TYPE "Invalid type" if the *ace_type* is not one of **LIST**, **RLIST**, **USER**, or **RUSER**; MR_NO_MATCH "No objects in database match query" if the *ace_name* doesn't match a user or list.

qualified_get_lists, qgli(*active, public, hidden, maillist, group***) =>** *list*

Finds the names of any lists that meet the specified criteria. Each of the inputs may be one of **TRUE, FALSE**, or **DONTCARE**. Any user may execute this query with *active* **TRUE** and *hidden* **FALSE**. Errors: MR_TYPE "Invalid type" if one of the arguments is something other than **TRUE**, **FALSE**, or **DONTCARE**.

get_members_of_list, gmol(*list***) =>** *type, value*

Retrieves all of the members of the named list. The *list* must match exactly one list in the database. The returned pairs consist of the type **USER**, **LIST**, or **STRING**, followed by the login name, list name, or text string respectively. This query may be executed by anyone if the list is not hidden; otherwise by someone on the ACE of the list being modified.

get_lists_of_member, glom(*type, value***) =>** *list, active, public, hidden, maillist, group*

Retrieves the name and flags of every list containing the named member. The member *type* must be one of **USER**, **LIST**, or **STRING**, and the *value* a login name, list name, or text string respectively. The *type* may also be one of **RUSER, RLIST**, or **RSTRING**, in which case it will also find any lists that contain as sublists a list that the target is a member of. This query may be executed by someone asking about themselves or a person on the ace of a list asking about that list. Errors: MR_TYPE "Invalid type" if the *type* is not **USER, LIST, STRING, RUSER, RLIST**, or **RSTRING**; or MR_NO_MATCH "No records in database match query" if *value* does not match an existing value for the given type.

count_members_of_list, cmol(*list***) =>** *count*

Determines how many members are on the specified list. The *list* name must match exactly one list. This query may be executed by a anyone on a visible list, or someone on the ACE of the target list. MR_LIST "Invalid list" if the list name does not match exactly one list.

DCM Information

get_server_info, gsin(*name***) =>** *service, interval, target, script, dfgen, dfcheck, type, enable, inprogress, harderror, errmsg, ace_type, ace_name, modtime, modby, modwith*

Retrieves service information from the database. This is the per-service information used by the DCM for updates. The service *name* may contain wildcards. Note that all service names are stored in uppercase, and the passed name will be upper-cased before comparing it. The returned *interval* is in minutes. *dfgen* is the data files were last generated, and *dfcheck* is the date we last checked to see if we needed to generate them. These are passed as integers (unix format date). The *type* must be a **service-type** as stored in the aliases database. *enable, inprogress*, and *harderror* are booleans (0 = false, non-zero = true). *ace_type* is either **USER, LIST**, or **NONE**, and *ace_name* is a login name, a list name, or **NONE**, respectively. This query may be executed by someone on the service ace if only one service is retrieved.

qualified_get_server, qgsv(*enable, inprogress, harderror***) =>** *service*

Finds the names of any services that meet the specified criteria. Each of the inputs may be one of **TRUE, FALSE**, or **DONTCARE**. Errors: MR_TYPE "Invalid type" if any of the flags are not one of the three legal values.

add_server_info, asin(*service, interval, target, script, type, enable, ace_type, ace_name***) =>**

Adds a new service to the database. This is the per-service information used by the DCM for updates. Note that only a subset of the information is added in this query, as the remaining fields are only changed by the DCM with the *set_server_internal_flags* query. The *service* name will be converted to uppercase. The *interval* is in minutes. The *type* must be a **service-type** as stored in the aliases database. *Enable* is a boolean (0 = false, non-zero = true). *ace_type* is either **USER, LIST**, or **NONE**, and *ace_name* is a login name, a list name, or **NONE**, respectively. The service modtime will be set. Errors: MR_TYPE "Invalid type" if the type field is not a valid **service-type** in the alias database, or MR_ACE "No such access control entity" if the *ace_type* is not **USER, LIST**, or **NONE** or the *ace_name* cannot be resolved based on the *ace_type*.

update_server_info, usin(*service, interval, target, script, type, enable, ace_type, ace_name***) =>**

Updates a service in the database. This is the per-service information used by the DCM for updates. Note that only a subset of the information can be modified with this query, as the remaining fields are only changed by Moira itself. The *service* name must match exactly one existing service after being converted to uppercase. The *interval* is in minutes. The *type* must be a **service-type** as stored in the aliases database. *Enable* is a boolean (0 = false, non-zero = true). *Ace_type* is either **USER, LIST**, or **NONE**, and *ace_name* is a login name, a list name, or **NONE**, respectively. The service modtime will be set. This query may be used by someone on the ACE of the target service. Errors: MR_TYPE "Invalid type" if the *type* field is not a valid **service-type** in the alias database, or MR_ACE "No such access control entity" if the *ace_type* is not **USER, LIST**, or **NONE** or the *ace_name* cannot be resolved based on the *ace_type*.

reset_server_error, rsve(*service***) =>**

Updates the specified service by changing the harderror flag from **TRUE** to **FALSE**, and sets *dfcheck* to be the same as *dfgen*. The *service* name must match exactly on existing service after being converted to uppercase. The service modtime will be set. This query may be executed by someone on the ACE of the target service.

set_server_internal_flags, ssif(*service, dfgen, dfcheck, inprogress, harderr, errmsg***) =>**

Updates the specified service. This is intended to only be used by the DCM, as it changes flags that the user should not have control over. The *service* name must match exactly one existing service after being converted to uppercase. *dfgen* and *dfcheck* are unix format dates (long integers). *inprogress* and *harderr* are booleans (0 = false, non-zero = true). The service modtime will NOT be set.

delete_server_info, dsin(*service***) =>**

Deletes a set of service information from the database. The *service* name must match exactly one service in the database after being converted to uppercase. A service may not be deleted if there are any server-hosts assigned to that service, or if the inprogress bit is set for that service. Error: MR_IN_USE "Object is in use" if there are hosts assigned to that service.

get_server_host_info, gshi(*service, machine***) =>** *service, machine, enable, override, success, inprogress, hosterror, errmsg, lasttry, lastsuccess, value1, value2, value3, modtime, modby, modwith*

Retrieves server-host information from the database. This is the per-host information used by the DCM for updates. The given *service* and *machine* names may contain wildcards. *Enable, override, success, inprogress*, and *hosterror* are booleans (0 = false, non-zero = true). *lasttry* and *lastsuccess* are unix format dates (long integers). This query may be executed by someone on the ACE for the target service.

qualified_get_server_host, qgsh(*service, enable, override, success, inprogress, hosterror***) =>** *service, machine*

Finds the names of any machine/services pairs that meet the specified criteria. The *service* name may contain wildcards. Each of the remaining inputs may be one of **TRUE, FALSE**, or **DONTCARE**. Errors: MR_TYPE "Invalid type" if any of the flags are not one of the three legal values.

add_server_host_info, ashi(*service, machine, enable, value1, value2, value3***) =>**

Adds information for a new server-host to the database. This is the per-host information used by the DCM for updates. Note that only a subset of the information is dealt with in this query, as the remaining fields are only changed by the DCM with the *set_server_host_internal* query. *Service* and *machine* must each match exactly one existing service and machine, respectively. *Enable* is a boolean (0 = false, non-zero = true). The 3 values are service specific in function; *value1* and *value2* are integers, *value3* is a string. The server-host's modtime will be set. This query may be used by someone on the ACE for the target service. Errors: MR_SERVICE "Unknown service" if the *service* name does not match exactly one existing service, or MR_MACH "Invalid machine" if the *machine* name does not match exactly one machine.

update_server_host_info, ushi(*service, machine, enable, value1, value2, value3***) =>**

Updates information for a server-host in the database. This is the per-host information used by the DCM for updates. Note that only a subset of the information is dealt with in this query, as the remaining fields are only changed by the DCM with the *set_server_host_internal* query. *Service* and *machine* must each match exactly one existing service and machine, respectively. *Enable* is a boolean (0 = false, non-zero = true). The 3 values are service specific in function; *value1* and *value2* are integers, *value3* is a string. The server-host's modtime will be set. This query may only be executed when the inprogress bit is not currently set for the specified server_host. This query may be used by someone on the ACE for the target service. Errors: MR_SERVICE "Unknown service" if the *service* name does not match exactly one existing service, or MR_MACH "Invalid machine" if the *machine* name does not match exactly one machine.

reset_server_host_error, rshe(*service, machine***) =>**

Resets the hosterr flag for the specified server_host. The *service* and *machine* must each match exactly one service and host. The server_host's modtime will be updated. This query may be used by someone on the ACE for the target service. Errors: MR_SERVICE "Unknown service" if the *service* name does not match exactly one existing service, or MR_MACH "Invalid machine" if the *machine* name does not match exactly one machine.

set_server_host_override, ssho(*service, machine***) =>**

This will set the override flag for a server_host, and start a new DCM running. The *service* and *machine* must each match exactly one service and host. The server_host's modtime will be updated. This query may be used by someone on the ACE for the target service. Errors: MR_SERVICE "Unknown service" if the *service* name does not match exactly one existing service, or MR_MACH "Invalid machine" if the *machine* name does not match exactly one machine.

set_server_host_internal, sshi(*service, machine, override, success, inprogress, hosterror, errmsg, lasttry, lastsuccess***) =>**

Updates the specified service_host. This is intended to only be used by the DCM, as it changes flags that the user should not have control over. The *service* and *host* names name must match exactly one existing service and host each. *override, success, inprogress* and *hosterror* are booleans (0 = false, non-zero = true). *lasttry* and *lastsuccess* are unix format dates (long integers). The service_host modtime will NOT be set. Errors: MR_SERVICE "Unknown service" if the *service* name does not match exactly one existing service, or MR_MACH "Invalid machine" if the *machine* name does not match exactly one machine

delete_server_host_info, dshi(*service, machine***) =>**

Deletes a server-host from the database. The *service* and *machine* names each must match exactly one existing service or host. A server-host may not be deleted if the inprogress bit is set for that server-host. This query may be used by someone on the ACE for the target service. Errors: MR_SERVICE "Unknown service" if the *service* name does not match exactly one existing service, MR_MACH "Invalid machine" if the *machine* name does not match exactly one machine, or MR_IN_USE "Object is in use" if the inprogress bit is set.

get_server_locations, gslo(*service***) =>** *service, machine*

This query tells which machines support a given service. It does this by listing each of the server-hosts for that service. The *service* name may contain wildcards, and will be converted to uppercase before any comparisons are made. It is safe for this query's ACL to be the list containing everybody.

Filesystems

get_filesys_by_label, gfsl(*name***) =>** *name, fstype, machine, packname, mountpoint, access, comments, owner, owners, create, lockertype, modtime, modby, modwith*

Retrieves all the information about a specific filesystem from the database. The *name* may contain wildcards. *fstype* is one of **NFS**, **RVD**, **FSGROUP**, or **ERR**, recorded as aliases of **filesys**. *machine* must match exactly one existing machine. *owner* must match exactly one user, *owners* must match exactly one list. *create* is a boolean (0 = false, non-zero = true) indicating that the locker should be automatically created. *lockertype* is a **lockertype** as recorded in the alias database, currently one of **SYSTEM, HOMEDIR, PROJECT**, or **OTHER**. The *packname, mountpoint*, and *access* vary depending on the filesystem type.

get_filesys_by_machine, gfsm(*machine***) =>** *name, fstype, machine, packname, mountpoint, access, comments, owner, owners, create, lockertype, modtime, modby, modwith*

Retrieves the information about any filesystems on the named machine. The *machine* name must match exactly one machine in the database. The returned information is as specified above for *get_filesys_by_label*. Errors: MR_MACHINE "No such machine" if the named machine does not match an existing machine.

get_filesys_by_nfsphys, gfsn(*machine, partition***) =>** *name, fstype, machine, packname, mountpoint, access, comments, owner, owners, create, lockertype, modtime, modby, modwith*

Retrieves the information about all NFS filesystems that reside on the specified NFS server partition. *machine* must match exactly one machine. *partition* is the mount point of the NFS physical partition. Errors: MR_MACHINE "Invalid machine" if the machine name does not match exactly one machine, or MR_NO_MATCH "No records in database match query" if the partition does not match anything.

get_filesys_by_group, gfsg(*list***) =>** *name, fstype, machine, packname, mountpoint, access, comments, owner, owners, create, lockertype, modtime, modby, modwith*

Retrieves the information about all filesystems that have the specified group as the owners list. The *list* must match exactly one existing list. This query may be executed by a member of the target list. Errors: MR_LIST "No such list" if the given list does not match exactly one list in the database.

add_filesys, afil(*name, fstype, machine, packname, mountpoint, access, comments, owner, owners, create, lockertype***) =>**

Adds a new filesystem to the database. The *name* must be unique among the existing filesystems. *fstype* is one of **NFS** or **RVD**. *machine* must match exactly one existing machine. *owner* must match exactly one user, *owners* must match exactly one list. *create* is a boolean (0 = false, non-zero = true) indicating that the locker should be automatically created. *lockertype* is a **lockertype** as recorded in the alias database, currently one of **SYSTEM, HOMEDIR, PROJECT**, or **OTHER**. The *packname* and *access* vary depending on the filesystem type. For an RVD filesystem, they may contain anything. For NFS filesystems, the *packname* must match an existing NFS physical filesystem, and *access* must be one of **r** or **w**. The filesystem's modtime will be set. Errors: MR_FSTYPE "Invalid filesys type" if the *fstype* is not a valid **filesys** type, MR_TYPE "Invalid type" if the *lockertype* is not a valid **lockertype**, MR_MACHINE "No such machine" if the *machine* name does not match exactly one machine, MR_USER "No such user" if the *owner* does not match exactly one user, MR_LIST "No such list" if the *owners* does not match exactly one list, MR_NFS "Specified directory not exported" if the *machine* and *packname* do not match an existing NFS physical partition, or MR_FILESYS_ACCESS if the *fstype* is **NFS** and the access mode is not **r** or **w**.

update_filesys, ufil(*name, newname, fstype, machine, packname, mountpoint, access, comments, owner, owners, create, lockertype***) =>**

Updates the information about a filesystem in the database. The *name* must match exactly one existing filesystem. The *new name* must either match the existing one or be unique among the filesystems. *fstype* is one of **NFS** or **RVD**. *machine* must match exactly one existing machine. *owner* must match exactly one user, *owners* must match exactly one list. *create* is a boolean (0 = false, non-zero = true) indicating that the locker should be automatically created. *lockertype* is a **lockertype** as recorded in the alias database, currently one of **SYSTEM, HOMEDIR, PROJECT**, or **OTHER**. The *packname* and *access* vary depending on the filesystem type. For an **RVD** filesystem, they may contain anything. For **NFS** filesystems, the *packname* must match an existing NFS physical filesystem, and *access* must be one of **r** or **w**. The filesystem's modtime will be updated. Errors: MR_NOT_UNIQUE "Arguments not unique" if the *new name* does not either match the old one or is unique among filesystems, MR_FSTYPE "Invalid filesys type" if the *fstype* is not a valid **filesys** type, MR_TYPE "Invalid type" if the *lockertype* is not a valid **lockertype**, MR_MACHINE "No such machine" if the *machine* name does not match exactly one machine, MR_USER "No such user" if the *owner* does not match exactly one user, MR_LIST "No such list" if the *owners* does not match exactly one list, MR_NFS "Specified directory not exported" if the *machine* and *packname* do not match an existing NFS physical partition, or MR_FILESYS_ACCESS if the *fstype* is **NFS** and the *access* mode is not **r** or **w**.

delete_filesys, dfil(*name***) =>**

Deletes a filesystem from the database. The *name* must match exactly one existing filesystem. Any quotas assigned to that filesystem will be deleted, and the allocation count on the nfs physical partition will be decremented accordingly. Errors: MR_FILESYS "No such file system" if the name does not match an existing filesystem.

get_fsgroup_members, gfgm(*name***) =>** *filsys, sortkey*

Fetches the filesystems belonging to the named fsgroup. The filesystems will be returned in sort order. The *name* must match exactly one existing fsgroup. Errors: MR_FILESYS "No such file system" if the name does not match an existing fsgroup.

add_filesys_to_fsgroup, aftg(*fsgroup, filesys, sortkey***) =>**

Adds the named filesystem to the named fsgroup, with the given sortkey. The *filesys* and *fsgroup* must each match exactly one existing filesystem. Errors: MR_FILESYS "No such file system" if the name does not match an existing fsgroup.

remove_filesys_from_fsgroup, rffg(*filesys, fsgroup***) =>**

Deletes the named filesystem from the named fsgroup. The *filesys* and *fsgroup* must each match exactly one existing filesystem. Errors: MR_FILESYS "No such file system" if the name does not match an existing fsgroup, MR_NO_MATCH if the filesystem is not a member of the group.

get_all_nfsphys, ganf() => *machine, dir, device, status, allocated, size, modtime, modby, modwith*
Retrieves information about NFS physical filesystems. These are the filesystems which are exported by NFS servers.

get_nfsphys, gnfp(*machine, dir***) =>** *machine, dir, device, status, allocated, size, modtime, modby, modwith*
Retrieves information about a specific NFS physical filesystem. The *machine* must match exactly one existing machine. The *dir*ectory name may contain wildcards. Errors: MR_MACHINE "No such machine" if the *machine* name does not match exactly one existing machine.

add_nfsphys, anfp(*machine, directory, device, status, allocated, size***) =>**
Adds a new NFS physical filesystem to the database. The *machine* name must match exactly one existing machine. The *directory* and *device* must be unique among existing NFS physical filesystems for this machine. *status* is an integer, with bit encodings **MR_FS_STUDENT, MR_FS_FACULTY, MR_FS_STAFF**, or **MR_FS_MISC** as defined in <*moira.h*>. *allocated* keeps track of quota allocation, the initial value should be zero unless there is something besides lockers on this filesystem. *size* is the actual size (in blocks) of the filesystem. The modtime will be set for this filesystem. Errors: MR_MACHINE "No such machine" if the *machine* name does not match exactly one existing machine.

update_nfsphys, unfp(*machine, directory, device, status, allocated, size***) =>**
Changes information about an NFS physical filesystem in the database. The *machine* name must match exactly one existing machine. The *directory* must match an existing NFS physical filesystem on that machine. The remaining arguments will replace the current values of those fields. The modtime will be updated for this filesystem. Errors: "No such machine" if the *machine* name does not match exactly one existing machine.

adjust_nfsphys_allocation, ajnf(*machine, directory, delta***) =>**
Changes the allocation for an NFS physical filesystem. *machine* must match exactly one existing machine. *directory* must match an existing NFS physical filesystem on that machine. The current allocation for this filesystem will have *delta* (which may be positive or negative) added to it. Errors: MR_MACHINE "No such machine" if the *machine* name does not match exactly one existing machine.

delete_nfsphys, dnfp(*machine, directory***) =>**
Deletes an NFS physical filesystem from the database. The *machine* name must match exactly one existing machine. The *directory* name must match exactly one existing NFS physical filesystem on that machine. The physical filesystem must not be in use with logical filesystems. Errors: MR_MACHINE "No such machine" if the *machine* name does not match exactly one existing machine, or MR_IN_USE "Object is in use" if there are any filesystems assigned to this partition.

get_quota, gqot(*filesys, type, name***) =>** *filesys, type, name, quota, directory, machine, modtime, modby, modwith*
Retrieves the quotas assigned to the named filesystem(s) and quota holder. Valid types are **USER**, **GROUP**, and **ANY** with corresponding login name, list name, or any placeholder. The *filesystem* name may contain wildcards. Errors: MR_TYPE "Invalid type" if the type is not a valid "quota_type" in the alias table.

get_quota_by_filesys, gqbf(*filesys***) =>** *filesys, type, name, quota, directory, machine, modtime, modby, modwith*
Retrieves the quotas assigned to the named filesystem(s). The *filesystem* name may contain wildcards.

add_quota, aqot(*filesystem, type, name, quota***) =>**
Adds a new quota to the database. The *filesystem* name must match exactly one existing filesystem. Valid types are **USER**, **GROUP**, and **ANY** with corresponding login name, list name, or "*". The *quota* may be any positive integer. The modtime on the quota record will be set. If the filesystem is of type NFS, the allocation count

for that NFS physical partition will also be updated. Errors: MR_FILESYS "No such file system" if the *filesystem* does not match exactly one existing filesystem, or MR_TYPE "Invalid type" if the type is not a valid "quota_type" in the alias table.

update_quota, uqot(*filesystem, type, name, quota***) =>**

Changes a quota in the database. The *filesystem* name must match exactly one existing filesystem. Valid types are **USER**, **GROUP**, and **ANY** with corresponding login name, list name, or "*", and that entity must have a quota on the named filesystem. The quota may be any positive integer, and will replace the existing quota. The modtime on the quota record will be set. If the filesystem is of type NFS, the allocation count for that NFS physical partition will also be updated. Errors: MR_FILESYS "No such file system" if the *filesystem* does not match exactly one existing filesystem, or MR_TYPE "Invalid type" if the type is not a valid "quota_type" in the alias table.

delete_quota, dqot(*filesystem, type, name***) =>**

Deletes a quota from the database. The *filesystem* name must match exactly one existing filesystem. Valid types are **USER**, **GROUP**, and **ANY** with corresponding login name, list name, or "*", and that entity must have a quota on the named filesystem. If the filesystem is of type NFS, the allocation count for that NFS physical partition will also be updated. Errors: MR_FILESYS "No such file system" if the *filesystem* does not match exactly one existing filesystem, or MR_TYPE "Invalid type" if the type is not a valid "quota_type" in the alias table.

get_nfs_quota, gnfq(*filesys, login***) =>** *filesys, login, quota, directory, machine, modtime, modby, modwith*

Obsolete. Retrieves the quotas assigned to the named filesystems and user. The *filesystem* name may contain wildcards. The *login* name must match exactly one user. This query may be executed by the owner of the target filesystem.

get_nfs_quotas_by_partition, gnqp(*machine, directory***) =>** *filesys, login, quota, directory, machine*

Retrieves the quotas assigned to a given device. The *machine* must match exactly one existing machine. The *directory* name may contain wildcards. Errors: MR_MACHINE "No such machine" if the *machine* name does not match exactly one existing machine.

add_nfs_quota, anfq(*filesystem, login, quota***) =>**

Obsolete. Adds a new quota to the database. The *filesystem* name must match exactly one existing filesystem. The *login* name must match exactly one existing user. The *quota* may be any positive integer. The modtime on the quota record will be set. The allocation count for that NFS physical filesystem will also be updated. Errors: MR_FILESYS "No such file system" if the *filesystem* does not match exactly one existing filesystem, or MR_USER "No such user" if the *login* name does not match exactly one existing user.

update_nfs_quota, unfq(*filesystem, login, quota***) =>**

Obsolete. Changes a quota in the database. The *filesystem* name must match exactly one existing filesystem. The *login* name must match exactly one existing user, and that user must have a quota assigned on that filesystem. The quota may be any positive integer, and will replace the existing quota. The modtime on the quota record will be set. The allocation count for that NFS physical filesystem will also be updated. Errors: MR_FILESYS "No such file system" if the *filesystem* does not match exactly one existing filesystem, or MR_USER "No such user" if the *login* name does not match exactly one existing user.

delete_nfs_quota, dnfq(*filesystem, login***) =>**

Obsolete. Deletes a quota from the database. The *filesystem* name must match exactly one existing filesystem. The *login* name must match exactly one existing user, and that user must have a quota assigned on that filesystem. The allocation count for that NFS physical filesystem will also be updated. Errors: MR_FILESYS "No such file system" if the *filesystem* does not match exactly one existing filesystem, or MR_USER "No such user" if the *login* name does not match exactly one existing user.

Zephyr

get_zephyr_class, gzcl(*class***) =>** *class, xmttype, xmtname, subtype, subname, iwstype, iwsname, iuitype, iuiname, modtime, modby, modwith*

Retrieves zephyr class information from the database. The *class* name may contain wildcards. There are four pairs of types and names: each type is one of **USER, LIST**, or **NONE**, and each name is a login name, a list name, or **NONE**, respectively.

add_zephyr_class, azcl(*class, xmttype, xmtname, subtype, subname, iwstype, iwsname, iuitype, iuiname***) =>**

Adds a new zephyr class to the database. The *class* name must be unique among the existing class names. There are four pairs of types and names: each type is one of **USER, LIST**, or **NONE**, and each name is a login name, a list name, or **NONE**, respectively. The class's modtime will be updated.

update_zephyr_class, uzcl(*class, newclass, xmttype, xmtname, subtype, subname, iwstype, iwsname, iuitype, iuiname***) =>**

Change a zephyr class in the database. The *class* name must match exactly one existing class. The *new class* name must either match the old one or be unique among the existing class names. There are four pairs of types and names: each type is one of **USER, LIST**, or **NONE**, and each name is a login name, a list name, or **NONE**, respectively. The class's modtime will be updated.

delete_zephyr_class, dzcl(*class***) =>**

Deletes a zephyr class from the database. The *class* name must match exactly one existing class.

Miscellaneous

get_server_host_access, gsha(*machine***) =>** *machine, ace_type, ace_name, modtime, modby, modwith*

Returns information about who has access to a given machine. This will be used to load the */.klogin* file on that machine. The *machine* name may contain wildcards. The *ace_type* is either **USER**, **LIST**, or **NONE**, and the *ace_name* is either a login name, a list name, or **NONE**, respectively.

add_server_host_access, asha(*machine, ace_type, ace_name***) =>**

Adds information about who has access to a given machine to the database. The *machine* name must match exactly one existing machine. The *ace_type* is either **USER, LIST**, or **NONE**, and the *ace_name* is either a login name, a list name, or **NONE**, respectively. The modtime on the record will be set. Errors: MR_MACHINE "No such machine" if the *machine* name does not match exactly one existing machine, MR_ACE "Invalid access control entity" if the *ace_type* and *ace_name* together do not specify a valid entity.

update_server_host_access, usha(*machine, ace_type, ace_name***) =>**

Updates the information about who has access to a given machine. The *machine* name must match exactly one existing machine. The *ace_type* is either **USER, LIST**, or **NONE**, and the *ace_name* is either a login name, a list name, or **NONE**, respectively. The modtime on the record will be updated. Errors: MR_MACHINE "No such machine" if the *machine* name does not match exactly one existing machine, MR_ACE "Invalid access control entity" if the *ace_type* and *ace_name* together do not specify a valid entity.

delete_server_host_access, dsha(*machine***) =>**

Updates the information about who has access to a given machine. The *machine* name must match exactly one existing machine.

get_printcap_entry, gpce(*printer***) =>** *printer, spool_host, spool_directory, rprinter, quotaserver, authenticate, price, comments, modtime, modby, modwith*

Retrieves information about a printer. The *printer* name may contain wildcards. It is safe for this query's ACL to be the list containing everybody.

add_printcap_entry, apce(*printer, spool_host, spool_directory, rprinter, quotaserver, authenticate, price,* *comments***) =>**

Adds information about a new printer to the database. The *printer* name must not match any existing printers. *spool_host* and *quotaserver* each must name exactly one existing machine in the database. The printer's modtime will be set. Error: MR_MACHINE if *spool_host* or *quotaserver* do not each match exactly one machine.

delete_printcap_entry, dpce(*printer***) =>**

Deletes information about a printer from the database. The *printer* name must match exactly one existing printer.

get_printcap, gpcp(*printer***) =>** *printer, spool_host, spool_directory, rprinter, comments, modtime, modby, modwith*

OBSOLETE. Retrieves information about a printer. The *printer* name may contain wildcards. It is safe for this query's ACL to be the list containing everybody.

add_printcap, apcp(*printer, spool_host, spool_directory, rprinter, comments***) =>**

OBSOLETE. Adds information about a new printer to the database. The *printer* name must not match any existing printers. *spool_host* must name exactly one existing machine in the database. The printer's modtime will be set. Error: MR_MACHINE if *spool_host* does not match exactly one machine.

delete_printcap, dpcp(*printer***) =>**

OBSOLETE. Deletes information about a printer from the database. The *printer* name must match exactly one existing printer.

get_palladium, gpdm(*name***) =>** *name, ident, machine, modtime, modby, modwith*

Gets information about palladium servers and supervisors. *Name* may contain wildcards.

add_palladium, apdm(*name, ident, machine***) =>**

Adds information about a new palladium server or supervisor. *Ident* is the 16-bit port identifier, and *machine* must name an existing machine. Error: MR_MACHINE if *machine* does not match exactly one machine.

delete_palladium, dpdm(*name***) =>**

Deletes information about a new palladium server or supervisor.

get_alias, gali(*name, type, translation***) =>** *name, type, translation*

Looks up an alias in the alias database. This database is used both for user information like alternate names of filesystems, and keyword validation for various queries. Note that type validation entries are of the form (*[type name, usually in lower case]*, **TYPE**, *[type string, always in upper case]*). Some type validation entries are used to further identify another field. These have entries of the form (*[type string in uppercase]*, **TYPEDATA**, *[type, one of:* **none, user, list, string, machine***]*). The *name, type*, and *translation* may contain wildcards. It is safe for this query to be the list containing everybody.

add_alias, aali(*name, type, translation***) =>**

Adds a new alias to the alias database. The *type* must be a known type as recorded under **alias** in the alias database. Duplicate translations for a given (name, type) pair are allowed. Note that type validation entries are of the form (*[type name, usually in lower case]*, **TYPE**, *[type string, always in upper case]*). Some type validation entries are used to further identify another field. These have entries of the form (*[type string in uppercase]*, **TYPEDATA**, *[type, one of:* **none, user, list, string, machine***]*). The *name, type*, and *translation* may contain wildcards.

delete_alias, dali(*name, type, translation***) =>**

Deletes an alias from the alias database. The combination of all three input arguments must match exactly one alias.

get_value, gval(*variable***) =>** *value*

Look up a value in the values database. This is used for DCM flags and Moira internal ID hints. The *variable* name must match exactly one existing value name in the database. It is safe for this query's ACL to be the list containing everybody. Errors: MR_NO_MATCH "No records in database match query" if the name does not match exactly one variable name.

add_value, aval(*variable, value***) =>**

Adds a new value to the values database. The *variable* name must be unique among the variables already in the database. The *value* is an integer.

update_value, uval(*variable, value***) =>**

Changes the value of an existing variable in the values database. The *variable* name must match exactly one existing variable. Its *value* will be replaced with the supplied value. Errors: MR_NO_MATCH "No records in database match query" if the name does not match exactly one variable name.

delete_value, dval(*variable***) =>**

Deletes a variable from the values database. The *variable* name must match exactly one existing variable. Errors: MR_NO_MATCH "No records in database match query" if the name does not match exactly one variable name.

get_all_table_stats, gats() => *table, retrieves, appends, updates, deletes, modtime*

Retrieves a summary of the table statistics. Each tuple consists of the *table* name, how many *retrieves, appends, updates*, and *deletes* have been performed on that table, and the date of the last change to the table. It is safe for this query's ACL to be the list containing everybody.

Built-in Special Queries

_help(*query***) =>** *help_message*

Returns the short name of the query and a list of arguments and return values. The query name must match an existing query. This query may be executed by anyone.

_list_queries() => *long_query_name, short_query_name*

Returns a list of every query name. This query may be executed by anyone.

_list_users() => *kerberos_principal, host_address, port_number, connect_time, client_number*

Returns a list of every client currently using the Moira server. This query may be executed by anyone.