To: Distribution

From: Barry Margolin

Date: August 17, 1982

Subject: The Multics Inquire System: A User-Accessible, User-Maintained, Personal User Database

1. ABSTRACT

The Multics Inquire system can provide answers to many of the Multics users' questions about the other users of the system, such as "what project is JFoo on, so I can send him mail?" or "is James Smith a user on this computer, and if so what is his userid?". Inquire is a system-wide user-accessible database of all the users of a Multics system, including information on each user that may be useful to other users. This database will contain computer-related information such as the user's computer mail address as well as non-computer-related data such as his home address. The information in the database is maintained by the users themselves, so no interaction with system administrators is required to inform the rest of the user community of one's change in address. The implementation described herein allows control of the Inquire information by allowing a user to selectively suppress the dissemination of particular data items to the general user community.

Throughout this document the name "Inquire" will be used to refer to this software. This is not expected to be the actual name of the product, as that name is currently a registered trademark for another vendor's software. When a name for this software is chosen some command names will be changed appropriately.

Send comments on this MTB by one of the following means:

By Multics Mail, on MIT or System M, to:
Margolin.Multics

To the forum meetings (method of choice):
Inquire-Development (inquire) in the MIT default meeting directory >udd>m>barmar>mtgs>Inquire-Development (inqdev) on System M

By telephone:
HVN 261-9321 or (617) 253-7788

2. INTRODUCTION

The Multics Inquire System is the combination of a user-maintained system-wide database of the users of a Multics installation and software to provide access to the the information to other users and system programs. For each user the database contains the full name, address, computer mailbox address, preferred eor options, and any other per-user information that is found to be useful. This concept is based upon the INQUIR system, originally designed for the PDP-10's at the MIT Laboratory for Computer Science (formerly Project MAC) and Artificial Intelligence Laboratory, and currently in operation on those systems and several others at MIT and around the country.

Other programs can make use of this information in useful ways. For example, the mail system will use the Inquire database as a "mail table," allowing mail addressed to "Margolin" to be delivered to "Margolin.PDO"'s mailbox based on the information in Margolin's Inquire entry. Automatic forwarding to networks will also be provided. The eor command could queue its output to the pps_2sided printer with a destination of "Palter" if I typed "eor output.file -for Palter" (1) and Palter's eor_args field specified those options. If I just wanted to find out CHoffman's home phone number, I could say "display_inquire_fields CHoffman -fields home_phone". Programs used by the site support groups might also make use of the Inquire System; one suggestion is a field which would tell the documentation group what documents should be sent to the user (at the address in the home_address field, of course) when they are updated.

Other features of the Inquire system relate to the security needs of the users. Since all users must be able to write into the database, there must be a way to prevent a user from writing into another user's entry. It also must be possible to prevent people from viewing data in the database that the user wishes kept private. This feature is intended to deal with the cases where information is automatically entered into the database without the user's explicit request. This will probably include the initial loading of the database by copying the information out of the URF; there are also plans for fields that are maintained automatically by the Inquire software, such as an entry modification date.

Some of the information that Inquire will be maintaining is currently managed in other ways on Multics. Full names and addresses are maintained in the User Registration File (URF); unfortunately, this is generally read-protected from the majority of the users of a system and write-protected from all but systems-maintenance users, so it is often out of date and incorrect. There is an ad hoc mechanism for routing mail without projects by looking for a link with that name (and a mbx suffix) in a central directory; this mechanism is generally only available to systems personnel and their friends, since only they can add entries to

(1) At this point in time, this syntax for eor is just an example. The actual control argument name will be decided upon later.

this directory.

3. IMPLEMENTATION

The important requirements of an implementation for the Inquire System are:

1) Efficient keying and searching. There will be many entries in this database (one for each user), and many programs will have to look up entries based upon characteristics of the entry (as opposed to keying on the userid or last name, which are primary and secondary database keys in the current implementation). I hope that many facets of the Multics system will use the database, and the database lookups should not be a system bottleneck.
2) Security via the ring protection mechanism. Since most users must have the access to write and read the database, ACL checking is not enough to maintain the proper level of security. Therefore, the ring mechanism must be employed to prevent incorrect database access. (If used at AIM sites, the database can be set at system_low in order to allow most users to access it.)
3) Extensibility. Different sites will have different needs that should be satisfied by the Inquire System. They must have the ability to customize the database to some extent, particularly the fields that are contained in the entries.
4) Robustness. The implementation should allow simultaneous accesses to the database, without locking people out unnecesarily. Future plans include some method of recording changes to the database, or at least preserving previous states and the ability to restore single entries if necessary.

The range of underlying data manager implementations I had to choose from was very limited: Multics Relational Data Store (MRDS), Relational Data Management System (RDMS), indexed vfile_, and an original database manager that I would have had to write. The last was rejected early in the planning stage, as the implementation of Inquire was originally intended to be a summer project, and such an undertaking requires many man-months. RDMS was also rejected, as it is an MIT installation-maintained product, and Inquire was meant to become a Honeywell standard product. Indexed vfile_ was my first choice, but due to the anticipated work involved in implementing the necessary multiple keying and database robustness in such an implementation, I decided to use MRDS, which handles efficient keying and database locking and consistency. (1) The only forseeable problem with this decision is the fact that MRDS is not as storage efficient as a direct vfile_ implementation could be. I have decided to accept this inefficiency for the time being in order to gain the modularity that MRDS will give me (the current implementation of Inquire is modularly designed to allow easy substitution of the low-level

(1) During implementation I discovered that MRDS' locking does not work properly in the inner ring when contention occurs, and had to wrap my own locks around the MRDS locks in order to prevent contention. When this deficiency is corrected, this code will be removed.

database manager).

4. DATABASE ORGANIZATION

The Inquire database is a collection of "entries", where each entry represents a single user of the Multics system (an extension to this definition will allow entries which represent non-user entities, such as mail distribution lists). Each of these entries contains a set of "fields" which contain information about the user, and each field has a fieldname. The contents of all fields are character string data. The Inquire system commonly looks up entries based upon the "userid" fields, so this field is the "key field", and lookups by userid will be highly efficient; because this is the primary key of the database, these fields must be unique among all the entries in the database, but Multics already requires this of userids. A secondary key on the entry is the last_name field, as I expect that a major use of the database (when not looking up information based on a userid) will be to find out what userid corresponds with a given person. The Multics user whose userid matches the userid field in an Inquire entry is considered the "owner" of that entry. Associated with every field will be a "private-flag," and each entry will also have an entry privacy-flag. The properties of the owner and the function of the private flag are discussed in the "SECURITY" section.

The Inquire database manager will not regulate the actual content of the database except for the userid field of every entry, which it maintains (and any other automated fields that may be added in the future -- see the "FUTURE ENHANCEMENTS" section) and uses for security purposes. Inquire will also not format particular fields in any way; as far as it is concerned, all field contents are just arbitrary character strings. Since certain fields are intended to be parsed by other programs (see the "INTERFACES" and "INITIAL CONTENT" sections below) there should be comands provided with those subsystems for filling in this data in the proper format. This format should, nonetheless, be at least readable by humans and, if possible, easily generated by humans; because there is nothing preventing a user from modifying the contents of these fields himself programs which use them should be prepared for unformatted text and either provide suitable diagnostics or use defaults (the latter is preferable, as the user who runs into the error is most likely not the owner of the faulty entry).

5. INITIAL CONTENT

During the course of the design of the Inquire System, approximately thirty to forty fields were suggested for inclusion in users' database entries. The initial implementation of Inquire will not attempt to support all these suggested fields. The following fields seemed to be considered the most useful and basic, so they will be used in the initial database:

Field	Intended use or contents
Userid	(primary key)

Last name	(secondary key)
First names	
Full name	(for printing out, including titles, suffixes, etc.)
Mail system address	(parsable by the mail system)
Home street address	
Home city	
Home state	(two-letter standard abbreviation in US)
Home country	
Home zip code	(if not in US, use for the equivalent postal code)
Home phone #	
Work company name	
Work street address	
Work city	
Work state	
Work country	
Work zip code	
Work phone #	
Birthday	(Without the year of birth)
EOR options	(value segment pathname)
Location	(user's current location or last logout time)
Remarks	(just about anything the user wishes to include)
Plan	(pathname of a "plan file", which contains info about how to find the person)

As stated in the "DATABASE ORGANIZATION" section, the actual contents of fields are not controlled by Inquire. Therefore, the use of the above fields can differ from the suggestions in the table. For example, outside the US the state fields can be used for the county or province as appropriate.

Sites will be able to customize and modify the set of fields that are contained in the Inquire database; however, no tools have been designed yet. Initial static customization is currently possible by modifying the cmdb segment (the source segment used to create a MRDS database) and a CDS segment that defines the relationship between Inquire field names and positions in the MRDS return structure. Dynamic modification has not yet been designed; MRDS restructuring will be available in MR10.1, although the proposed functionality does not seem to fill the needs of Inquire: there is no subroutine interface, and the ability to add attributes to relations is not included (the latter is planned for MR10.2). An alternate plan is to re-implement the Inquire System on a different database manager which supports dynamic restructuring. What I expect I will do is to include a large set of unused fields in the MRDS database, allowing for expansion of the number of accessible fields as they are added. Prior to these changes, Inquire restructuring will involve unloading and loading the database. A problem with most of the implementations is making sure that the software that is being used is consistent with the state of the database; the current implementation of the Inquire interface is poor in this respect, thus requiring that the database and associated software only be modified while the system is unavailable, otherwise there may be users running with an old version of the interface software and a new version of the database format.

A problem related to the initial content of the database is that of how to fill the database. Currently, much of the information that the Inquire database stores can be found in the User Registration File (URF); therefore, one possible approach is to copy the relevant parts of URF entries into the corresponding Inquire entries. Questions have been raised as to what the initial privacy on the entry should be and how to go about setting it properly; perhaps the MOTD facility could be used to suggest that users who wish their Inquire data to be public or private (whichever is not the default) to use the set_user_entry command in order to fix this (it seems more proper to default to private, although there has been some confusion by users of the initial implementation at MIT regarding this "feature"). Another approach is to put up a MOTD suggesting to users that they run the Inquire program in order to enter the initial data for themselves. The site may choose either of these methods; a tool to copy the URF information into the database is provided for those sites that choose this mechanism.

6. SECURITY

In this implementation, all database security will be maintained by the Inquire system software using the Multics ring mechanism for protection; the database will be kept in ring 2, so that it may only be accessed via the Inquire gates. There are plans to add similar security features to MRDS itself sometime in the future, and at that time Inquire may be modified to use these features if they provide the needed functionality. For those sites using the Access Isolation Mechanism (AIM), I suggest that the database be kept at system_low, so that most users may access the database; in any case, the database may only be accessed by users at the authorization level of the database, as the lock segments are written and read for both read and write lookups. In order for Inquire to provide more functionality on AIM sites, support will have to be provided in several other system routines, such as MRDS, vfile_, and msf_manager_, so that the Inquire database can be multi-class segments.

The owner of an entry has complete control over the contents of that database and whether other users may view this data. Write access in the Inquire database is simple: only the owner of an entry may modify that entry. Read access is controlled by the private-flags in that entry, which only the owner of an entry may update. As read access is somewhat more complicated than write access, a complete description of this mechanism follows.

For each field in the user's entry, there is a private-flag. If this flag is set, then that field will not be viewable when the database is being accessed by standard means; it will appear to anyone but the owner of the entry to be empty. It is expected that these flags will not be used very much. The best way to make most data private is to not enter it in the first place. At some point Inquire may be enhanced to automatically maintain some fields (such as the date/time the entry was last modified) rather than allowing the user to update them; the field-privacy flag mechanism will be one way to prevent other users from viewing these fields.

If it is found that these flags are hardly being used, they can easily be removed.

A private-entry flag will also be available. If this flag is set, the entire entry will apparently not be in the database at all. The purpose of this is to allow a user to turn off an entry in one step, rather than go through a long procedure to set each field to null (if a simple interface to such a procedure is developed, then this flag could be removed).

A set of privileged entrypoints to the Inquire subroutines are provided. These duplicate all the functionality of the standard entrypoints, but ignore the security features described above. Using these entrypoints system administrators may load the initial database, enter new users into the database (this is not necessary, as the user may create his own entry if he wishes), purge deleted users, and control offensive use of the database if necessary.

7. INTERFACES

As is the case with many Multics packages, there will be two levels of interface to the Inquire System: command/subsystem level and subroutine level. The subroutine interfaces are meant to be incorporated into many existing commands (e.g. who, eor) in order to extend their usefulness (if Inquire is distributed as a PSP, then the installed commands that use Inquire will need to check for its existence and provide appropriate diagnostics if it does not). User programs may also wish to make use of the data in the Inquire database, especially once its contents are expanded. The command and subsystem interfaces to Inquire will deal mainly with database maintenance, such as changing an address, and looking up a user. Although little system software will use the Inquire database initially, I expect that in time most accesses of the database will be made by non-Inquire software (consider the new meaning of "who -long", which could list full names along with the other information it gives) rather than via the Inquire commands. The next level of interface to the database will be an ssu_-based database maintenance subsystem, which would make it easy for a user to edit his entry and view other users' entries. Finally, several commands and active functions will be made available in order to make it easy for exec_coms to manipulate the information in the Inquire database. Some suggested improvements to MRDS (not expected until at least Multics MR11.0, if ever) involving built-in security may make it possible for users to access the Inquire database via standard MRDS interfaces, such as the dsl_ subroutine, mrds_call, and LINUS. The major command-level interfaces to Inquire will be the "display_inquire_fields" command/AF and the "set_user_entry" command, the the subroutine interface will be the "inquire_" subroutine, and the subsystem interface will be the "inquire" command; these will be described in the MPM-style documentation which follows.

The following existing facilities are expected to be modified in order to use the Inquire system: The mail system will be using it as its "mail table" in order to deliver and forward mail addressed without a project, and there are expected to be commands provided to manipulate the contents of

the field that is used; it may also put the full name from the Inquire entry into the header of the message; the enter_output_request command will be extended to allow the user to specify for whom the output is intended, and the appropriate control argument defaults will be taken from that user's Inquire entry; the logout command could be extended to (optionally) put the date/time in the user's Inquire entry. Finally, the birthday field could be used by a system daemon that would send a "birthday card" to users on their birthday, or even list the users whose birthday it is in the Multics herald (both these functions are performed on other systems with Inquire databases).

Since standard system modules are going to be using Inquire, it should not be a PSP item.

The currently planned interfaces will assume one system Inquire database. After the initial implementation, a set of entrypoints in which the location of the database to use is set may be created. This would allow users to set up private Inquire databases for whatever purposes they wish, using the interfaces provided by the Inquire system (although, for small applications without the need for write-protection mechanisms, a shared value segment would probably be a better approach). This would also allow the site to set up additional Inquire databases; a database of projects has been suggested. Because this requires the ability of the site to create MRDS databases, this capability depends upon whether the site has purchased the full MRDS package, which is a PSP, in addition to the retrieve/update MRDS interfaces, which will be in the standard system as of MR10.1.

8. FUTURE ENHANCEMENTS

The implementation so far described is adequate for many purposes, but it is not as complete as it could be. Some features that should be added include:

- o Read-only fields: The Inquire software could maintain various internal fields, such as the date of last update of an entry.

- o Case-insensitive matching: The user should be able to address mail to Margolin as "To: margolin" (the current add hoc mail table is searched without regard to case). This is a problem. Making the matching of userids case-insensitive could cause some requests to return multiple entries, even though the userid is supposed to be a unique identifier, but only with case taken into account.

- o Field name synonyms: The user should be able to refer to the office_address by a shorter name. The currently proposed implementation has exactly one name per field.

- o Field documentation: Each field in the database should be accompanied by a documentation string. This should describe the expected format of the field contents and the projected use of the field. This would be printed if the user types "?" to a field contents prompt, or uses

the describe_field inquire request.

- o Menu interface: A simple interface that would make it easy for novice users to update their Inquire entries and look at other users' entries.

- o Multiple databases: The ability to have more than one Inquire-style database. A database of projects has been suggested, and user-defined databases are also desireable. These have the problem that the per-entry access control issues are much harder to define (i.e., who may update the entry for a project).

- o Backup: Normal system backups are likely to catch the database in an inconsistent state. Some mechanism should be provided to backup a consistent database. This will probably be implemented as a privileged command that will prevent readers from gaining access to the database while it is being backed up. This backup would just copy the database hierarchy to another hierarchy, allowing it to be backed up by normal procedures.

- o Journalization: If the system crashes while the database is being modified the database could be left in an inconsistent state. Some updates would be lost if the database were restored from a backup. A journal file would allow the database to be brought up-to-date after such a crash. When MRDS is converted to the new data management system, this should be less of a problem, as this is dealt with by the DMS primitives.

- o Dynamic restructuring: System administrators should be able to add new fields, delete existing fields, or rename fields while the system is running.

- o Improved selection: The ability to retrieve entries based on the contents of fields other than the userid and the last name.

9. SCHEDULE

As of the time of this writing, FW229, the inquire_ subroutines and the display_inquire_fields command as documented in this MTB are working, although interfaces to several inquire_ subroutines differ from the documentation in this MTB. Manpower estimates for remainder of project:

inquire_ revision	2 man-weeks
inquire subsystem	4 man-weeks
other commands	2 man-weeks
testing	2 man-weeks
Total:	10 man-weeks

10. SYSTEM DEPENDENCIES

Inquire is expected to be released with MR10.1; however, for those who wish to install it on earlier systems, the following software is required:

MR10.0 ssu_
MR9.0 MRDS

11. DOCUMENTATION

The remainder of this document consists of MPM-style documentation for the interfaces to the Inquire System. The first section describes the inquire_ subroutine, and it will be followed by documentation of display_inquire_fields and related commands and the inquire subsystem.

Name: inquire_

The inquire_ subroutine provides an interface to the Inquire Personal User Database Manager.

Entry: inquire_$close_db

Function: Close the Inquire database, which is normally kept open for the entire process for efficiency. Use of this entry should not usually be necessary. The inquire_$priv_close_db entrypoint is also provided for parallelism, although it is exactly the same as the normal entrypoint (except that its use requires access to the inquire_priv_ gate).

Syntax:
```
declare inquire_$close_db entry (fixed bin (35));
call inquire_$close_db (code);
```

Arguments:
code
is a standard system error code or a code in inquire_et_. (Output)

Entry: inquire_$close_db_no_code

Function: The same as the close_db entrypoint, except that no error code is returned. This is currently provided so that it may be used by the Inquire system as an epilogue procedure that closes the database when the user logs out, since no arguments are passed to epilogue handlers. The inquire_$priv_close_db_no_code entrypoint is also provided for parallelism, although it is exactly the same as the normal entrypoint (except that its use requires access to the inquire_priv_ gate).

Syntax:
```
declare inquire_$close_db_no_code entry ();
call inquire_$close_db_no_code ();
```

Entry: inquire_$delete_entry

Function: This entrypoint removes the caller's entry from the Inquire database.

Syntax:
```
declare inquire_$delete_entry entry (fixed bin (35));
call inquire_$delete_entry (code);
```

Arguments:

code
is a standard system error code or a code in inquire_et_. inquire_et_$no_entry will be returned if the user is not currently in the Inquire database.

Entry: inquire_$get_all_userids

Function: Returns the userid fields of every non-private entry in the Inquire database.

Syntax:
```
declare inquire_$get_all_userids entry (ptr, char (8), ptr, fixed bin,
      fixed bin (35));
call inquire_$get_all_userids (area_ptr, userids_array_version,
      userids_ptr, userid_count, code);
```

Arguments:

area_ptr
is a pointer to an area in which the userids will be allocated. If it is null, then no userids will be returned, although the userid_count will be set. (Input)

userids_array_version
is the version of inq_userid_array that should be returned. The current version is the value of the variable inq_userids_array_version_1, declared in inquire_dcls.incl.pl1. This argument is ignored if area_ptr is null. (Input)

userids_ptr
is a pointer to the returned inq_userid_array, as declared in inquire_dcls. This will not be set if the area_ptr is null. (Output)

userid_count
is the number of elements in the userid array pointed to by userids_ptr, or that would be returned had the area_ptr not been null. (Output)

code
is a standard system error code or a code in inquire_et_. (Output)

Entry: inquire_$get_entry_privacy_flag

Function: Returns the value of the privacy flag for the caller's entry.

Syntax:
```
declare inquire_$get_entry_privacy_flag entry (bit (1), fixed bin (35));
call inquire_$get_entry_privacy_flag (privacy_flag, code);
```

Arguments:
privacy_flag
The value of the privacy flag setting of the caller's entry. If the value is "1"b, then the entry is private. (Output)
code
A standard system error code or a code in inquire_et_. (Output)

Entry: inquire_$get_field_names

Function: Returns the names of the fields in an Inquire entry.

Syntax:
```
declare inquire_$get_field_names entry (ptr, fixed bin (35));
call inquire_$get_field_names (field_names_ptr, code);
```

Arguments:
field_names_ptr
is a pointer to an inq_field_names structure, as declared in inquire_dcls.incl.pl1. The version component of this structure must be filled in prior to calling; this is the version of the structure that will be returned. Currently the only supported version is the value of the variable inq_field_names_version_1, also declared in the include file. The name component should be allocated with an extent of inquire_data_$field_count. (Input)
code
is a standard error code or a code in inquire_et_. (Output)

Entry: inquire_$get_field_privacy_flags

Function: Returns the privacy flag settings for the given fields in

inquire_ get_field_privacy_flags

the caller's entry.

Syntax:
```
declare inquire_$get_field_privacy_flags entry (ptr, ptr,
      fixed bin (35));
call inquire_$get_field_privacy_flags (field_names_ptr,
      field_privacies_ptr, bad_fields_ptr, code);
```

Arguments:

field_names_ptr
is a pointer to a standard inq_field_names structure, as declared in inquire_dcls.incl.pl1. This specifies the set of fields whose privacy flags will be returned. (Input)

field_privacies_ptr
is a pointer to a standard inq_field_privacies structure, as declared in inquire_dcls.incl.pl1. It will be filled in with the values of the privacy flags of the specified fields, in the order in which they were specified in inq_field_names. The version field should be filled in with the version of the structure expected; currently the only supported version is the value of the variable inq_field_privacies_version, also declared in inquire_dcls.incl.pl1. (Input)

bad_fields_ptr
is a pointer to to an inq_bad_fields structure, as declared in inquire_dcls.incl.pl1. This contains an array of bit(1) values which are set to "1"b iff the corresponding element of inq_field_names.name is an invalid field. This will only be filled in when code is inquire_et_$invalid_field. If this is null then this information will not be returned, but the error code will still be set. (Input)

code
is a standard system error code or a code in inquire_et_. (Output)

Notes: In the inq_field_privacies structure, a value of "1"b for a privacy flag signifies that the field is private; a "0"b value means it is public.

Entry: inquire_$get_fields_from_lname

Function: This entry retrieves the values of a given set of fields from all the entries whose last_name field matches the specified last name. If a particular field's privacy switch is set, a null string is returned for that value, unless the entry is the caller's own entry.

inquire_
get_fields_from_lname

inquire_
get_fields_from_lname

Syntax:
```
declare inquire_$get_fields_from_lname entry (char (*) var, ptr, ptr,
      char (8), ptr, ptr, fixed bin (35));
call inquire_$get_fields_from_lname (last_name, field_names_ptr,
      area_ptr, field_values_version, field_values_ptr, bad_fields_ptr,
      code);
```

Arguments:

last_name
: is the last name to be matched against in the database. This match is performed without regard to alphabetic case. (Input)

field_names_ptr
: is a pointer to a standard inq_field_names structure, as declared in inquire_dcls.incl.pl1. This specifies the set of fields whose values are to be returned. (Input)

area_ptr
: is a pointer to an area in which the return structure will be allocated. If it is null, the system area will be used. (Input)

field_values_version
: is the version of the inq_field_values structure the caller wishes returned. The current version is the value of the variable inq_field_values_version_1, declared in inquire_dcls.incl.pl1. (Input)

field_values_ptr
: is a pointer to a standard inq_field_values structure, as declared in inquire_dcls.incl.pl1, if the retrieval was successful. This contains the values found in the selected fields of the specified users' entries, in the same order as specified in inq_field_names. (Output)

bad_fields_ptr
: points to an inq_bad_fields structure. This contains an array of bit(1) values which are set to "1"b iff the corresponding element of inq_field_names.name is an invalid field. If code is inquire_et_$invalid_field and this pointer is non-null, then this structure will be filled in. (Input)

code
: is a standard system error code or a code in inquire_et_. (Output)

Entry: inquire_$get_fields_from_userid

Function: Returns the values of the specified set of fields in the given user's Inquire entry. If a particular field's privacy switch is set, a null string is returned for that value, unless the entry is the caller's own entry.

inquire_
get_fields_from_userid

inquire_
get_fields_from_userid

Syntax:
```
declare inquire_$get_fields_from_userid entry (char (*) var, ptr, ptr,
      char (8), ptr, ptr, fixed bin (35));
call inquire_$get_fields_from_userid (userid, field_names_ptr,
      area_ptr, field_values_version, field_values_ptr, bad_fields_ptr,
      code);
```

Arguments:

userid
: is the userid of the user whose data is being requested. (Input)

field_names_ptr
: is a pointer to a standard inq_field_names structure, as declared in inquire_dcls.incl.pl1. This specifies the set of fields whose values are to be returned. (Input)

case_sensitive
: if this parameter is "1"b, then the matching of userids with the database will be case-sensitive. If it is "0"b, then this matching will be done case-insensitively.

area_ptr
: is a pointer to an area in which the return structure will be allocated. If it is null then the system area will be used. (Input)

field_values_ptr
: is a pointer to a standard inq_field_values structure, as declared in inquire_dcls.incl.pl1, if the retrieval was successful. This contains the values found in the selected fields of the specified user's entry, in the same order as specified in inq_field_names. (Output)

bad_fields_ptr
: points to an inq_bad_fields structure. This contains an array of bit(1) values which are set to "1"b iff the corresponding element of inq_field_names.name is an invalid field. If code is inquire_et_$invalid_field and this pointer is non-null, then this structure will be filled in. (Input)

code
: is a standard system error code or a code in inquire_et_. (Output)

Notes: Because userids are unique in the Inquire database, there will only be one element in the entry array if the case_sensitive flag is "1"b.

Entry: inquire_$priv_delete_entry

Function: This entrypoint removes a specified entry from the Inquire

database. Use of this entrypoint requires access to the inquire_priv_ gate.

Syntax:
```
declare inquire_$priv_delete_entry entry (char (*) var, fixed bin (35));
call inquire_$priv_delete_entry (userid, code);
```

Arguments:
userid
is the userid of the entry which is to be deleted.
code
is a standard system error code or a code inquire_et_. inquire_et_$no_entry will be returned if the entry is not in the Inquire database.

Entry: inquire_$priv_get_all_userids

Function: Returns the userid fields of every entry in the Inquire database. These include entries with their private flags set, so use of this entrypoint requires access to the gate inquire_priv_.

Syntax and arguments are the same as for inquire_$get_all_userids.

Entry: inquire_$priv_get_entry_privacy_flag

Function: Returns the value of the privacy flag for the specified entry. Use of this entrypoint requires access to the gate inquire_priv_.

Syntax:
```
declare inquire_$priv_get_entry_privacy_flag entry (char (*) var,
      bit (1), fixed bin (35));
call inquire_$priv_get_entry_privacy_flag (userid, privacy_flag, code);
```

Arguments:
userid
is the userid of the entry whose privacy flag value is to be returned.
Other arguments are as for inquire_$get_entry_privacy_flag

Entry: inquire_$priv_get_field_privacy_flags

Function: Returns the privacy flag settings for the given fields in a specified entry. Use of this entrypoint requires access to the gate inquire_priv_.

Syntax:
```
declare inquire_$priv_get_field_privacy_flags entry (char (*) var, ptr,
     ptr, ptr, fixed bin (35));
call inquire_$priv_get_field_privacy_flags (userid, field_names_ptr,
     field_privacies_ptr, bad_fields_ptr, code);
```

Arguments:
userid
   is the userid of the entry whose field-privacy flags are to be modified.
Other arguments and notes are as for inquire_$get_field_privacy_flags.

Entry: inquire_$priv_get_fields_from_lname

Function: This entry retrieves the values of a given set of fields from all the entries whose last_name field matches the specified last name. This entrypoint ignores the privacy flags and thus requires access to the gate inquire_priv_.

Syntax and arguments are as for inquire_$get_fields_from_lname.

Entry: inquire_$priv_get_fields_from_userid

Function: Returns the values of the specified set of fields in the given user's Inquire entry. This entrypoint ignores all privacy flags and thus requires access to the gate inquire_priv_.

Syntax, arguments, and notes are as for inquire_$get_fields_from_userid.

inquire_
priv_set_entry_privacy_flag

Entry: inquire_$priv_set_entry_privacy_flag

Function: Sets the value of the privacy flag for a specified user's entry. If the entry did not previously exist it will be created, with all the fields set to the null string and all the field privacy flags set to "0"b (public). Use of this entrypoint requires access to the gate inquire_priv_.

Syntax:
```
declare inquire_$priv_set_entry_privacy_flag entry (char (*) var,
     bit (1), fixed bin (35));
call inquire_$priv_set_entry_privacy_flag (userid, privacy_flag, code);
```

Arguments:
userid
   is the userid of the entry whose privacy flag is to be modified.
Other arguments are as for inquire_$set_entry_privacy_flag.

Entry: inquire_$priv_set_field_privacy_flags

Function: Sets the privacy flags for the given fields in the a specified entry. If the entry did not previously exist it will be created, with all the fields set to the null string and the entry privacy flag and all unspecified field privacy flags set to "0"b (public). Use of this entrypoint requires access to the gate inquire_priv_.

Syntax:
```
declare inquire_$priv_set_field_privacy_flags entry (char (*) var, ptr,
     ptr, ptr, fixed bin (35));
call inquire_$priv_set_field_privacy_flags (userid, field_names_ptr,
     field_privacies_ptr, bad_fields_ptr, code);
```

Arguments:
userid
   is the userid of the entry whose field privacy flags are being modified.
Other arguments and notes are as for inquire_$set_field_privacy_flags.

Entry: inquire_$priv_set_fields

Function: Replaces the specified fields' values in the specified entry. If the entry did not previously exist it will be created, with all unspecified fields set to the null string and all privacy flags set to "0"b (public). Use of this entrypoint requires access to the gate inquire_priv_.

Syntax:
```
declare inquire_$priv_set_fields entry (char (*) var, ptr, ptr, ptr,
      fixed bin (35));
call inquire_$priv_set_fields (userid, field_names_ptr,
      field_values_ptr, bad_fields_ptr, code);
```

Arguments:
userid
   is the userid of the entry whose fields are to be set.
Other arguments and notes are as for inquire_$set_fields

Entry: inquire_$set_entry_privacy_flag

Function: Sets the value of the privacy flag for the caller's entry. If the entry did not previously exist it will be created, setting all fields to the null string and setting the field privacy flags to "0"b (public).

Syntax:
```
declare inquire_$set_entry_privacy_flag entry (bit (1),
      fixed bin (35));
call inquire_$set_entry_privacy_flag (userid, privacy_flag, code);
```

Arguments:
privacy_flag
   The new value of the privacy flag setting of the caller's entry. If the value is "1"b, then the entry is private. (Input)
code
   A standard system error code or a code in inquire_et_. (Output)

Entry: inquire_$set_field_privacy_flags

Function: Sets the privacy flags for the given fields in the caller's entry. If the entry did not previously exist it will be created, with all the fields set to the null string and the entry privacy flag and all unspecified field privacy flags set to "0"b (public).

Syntax:
```
declare inquire_$set_field_privacy_flags entry (ptr, ptr, ptr,
     fixed bin (35));
call inquire_$set_field_privacy_flags (field_names_ptr,
     field_privacies_ptr, bad_fields_ptr, code);
```

Arguments:

field_names_ptr
is a pointer to a standard inq_field_names structure, as declared in inquire_dcls.incl.pl1. This specifies the set of fields whose privacy flags will be changed. (Input)

field_privacies_ptr
is a pointer to a standard inq_field_privacies structure, as declared in inquire_dcls.incl.pl1. It contains the new values of the privacy flags of the specified fields, in the order in which they were specified in inq_field_names. (Output)

bad_fields_ptr
points to an inq_bad_fields structure. This contains an array of bit(1) values which are set to "1"b iff the corresponding element of inq_field_names.name is an invalid field. If code is inquire_et_$invalid_field and this pointer is non-null, then this structure will be filled in. (Input)

code
is a standard system error code or a code in inquire_et_. (Output)

Notes: In the inq_field_privacies structure, a value of "1"b for a privacy flag signifies that the field is private; a "0"b value means it is public.

Entry: inquire_$set_fields

Function: Replaces the specified fields' values in the caller's entry. If the entry did not previously exist it will be created, with all unspecified fields set to the null string and all privacy flags set to

"0"b (public).

Syntax:
```
declare inquire_$set_fields entry (ptr, ptr, ptr, fixed bin (35));
call inquire_$set_fields (field_names_ptr, field_values_ptr,
     bad_fields_ptr, code);
```

Arguments:

field_names_ptr
: is a pointer to a standard inq_field_names structure, as declared in inquire_dcls.incl.pl1. This specifies the set of fields whose values are to be modified. (Input)

field_values_ptr
: is a pointer to a standard inq_field_values structure, as declared in inquire_dcls.incl.pl1. This contains the new values to be stored into the user's entry, in the same order as specified in inq_field_names. (Input)

bad_fields_ptr
: points to an inq_bad_fields structure. This contains an array of bit(1) values which are set to "1"b iff the corresponding element of inq_field_names.name is an invalid field. If code is inquire_et_$invalid_field and this pointer is non-null, then this structure will be filled in. (Input)

code
: is a standard system error code or a code in inquire_et_. (Output)

Notes: If the user does not have an entry, one will be created, and the specified fields will be filled in. The entry thus created will have its privacy flag on, but the privacy switches of the fields will all be off. Any unspecified fields will contain the null string.

Name: inquire_data_

There are several useful data values in inquire_data_ data segment.

Entry: inquire_data_$field_count

Function: This is the number of fields in each entry of the inquire database.

Usage:
declare inquire_data_$field_count fixed bin external static;

Name: inquire_dcls.incl.pl1

The inquire_dcls include file contains the declarations for the structures used as arguments to several of the Inquire subroutines.

```
dcl     1 inq_field_names       based (inq_field_names_ptr) aligned,
          2 version             char (8),
          2 name_count          fixed bin (17),
          2 name                (inq_max_field_count
                                refer (inq_field_names.name_count))
                                char (32);
dcl     inq_field_names_ptr     ptr;
dcl     inq_field_names_version_1
                                char (8) int static options (constant)
                                init ("inqfn_01");

dcl     1 inq_field_values      based (inq_field_values_ptr) aligned,
          2 version             char (8),
          2 entry_count         fixed bin (17),
          2 value_count         fixed bin (17),
          2 entry               (inq_fv_size
                                refer (inq_field_values.entry_count)),
            3 value             (inq_max_field_count
                                refer (inq_field_values.value_count))
                                char (200) varying;
dcl     inq_field_values_ptr    pointer;
dcl     inq_field_values_version_1
                                char (8) int static options (constant)
                                init ("inqfv_01");
dcl     inq_fv_size             fixed bin (17);

dcl     1 inq_field_privacies   based (inq_field_privacies_ptr) aligned,
          2 version             char (8),
          2 value_count         fixed bin (17),
          2 value               (inq_max_field_count
                                refer (inq_field_privacies.value_count))
                                bit (1);
dcl     inq_field_privacies_ptr
                                ptr;
dcl     inq_field_privacies_version_1
                                char (8) int static options (constant)
                                init ("inqfp_01");

dcl     1 inq_bad_fields        based (inq_bad_fields_ptr) aligned,
          2 version             char (8),
          2 field               (inq_max_field_count)
                                bit (1) unaligned;
```

```
dcl     inq_bad_fields_ptr      ptr;
dcl     inq_bad_fields_version_1 char (8) int static
            options (constant) init ("inqbf_01");

dcl     inq_max_field_count     fixed bin init (30);        /* Just in case the progr
                                                            /* to set this, give a me

dcl     1 inq_userids_array     based (inq_userids_array_ptr) aligned,
          2 version             char (8),
          2 userids             (inq_userids_count) char (20);
dcl     inq_userids_array_ptr   ptr;
dcl     inq_userids_count       fixed bin;
dcl     inq_userids_array_version_1
                                char (8) int static options (constant)
                                init ("inqua_01");

dcl     inquire_data_$field_count
                                fixed bin external;
```

Name: delete_inquire_entry, dliqe

The delete_inquire_entry command allows the user to remove an entire entry from the Inquire database. The user is queried before the entries are deleted. If non-existent entries are specified the user will be notified, but the existing entries that were specified will also be deleted.

Syntax: dliqe {entry_specs} {-control_args}

Arguments:

entry_specs
Specify which entries should be removed from the database (see the section "Notes on entry_specs"). To remove entries other than one's own requires access to the gate inquire_priv_.

Control Arguments

-all, -a
Deletes all entries in the Inquire database. No entry_specs may be given if this control argument is specified.

-force, -fc
Deletes all the specified entries without querying the user.

-name userid, -nm userid
Used to specify a userid in an entry_spec if it begins with a hyphen. See the section "Notes on entry_specs".

-self
Used to specify the calling user in an entry_spec. See the section "Notes on entry_specs".

Notes:
At least one entry must be specified by arguments or control arguments.

Notes on entry_specs:
Entry_specs are used to specify an Inquire database entry to be deleted. Usually an entry_spec is just a userid, and the entry specified is the entry belonging to that user (by virtue of the specified userid matching the userid field of the entry). An entry_spec of -self can be used to specify the caller's entry. In

order to specify a userid that begins with a hyphen the userid should be preceded by the -name control argument.

Name: exists_inquire_entry, eiqe

The exists_inquire_entry command allows the user to determine if a particular entry exists in the Inquire database. This is useful when deciding whether to ask a user to fill in his or her user data.

Syntax: eiqe {entry_spec} {-control_args}

Syntax as an active function: [eiqe {entry_spec} {-control_args}]

Arguments:

entry_spec
Specifies the Inquire database entry whose existence is being checked (see the section "Notes on entry_specs"). One entry must be specified.

Control arguments:

-name userid, -nm userid
Used to specify a userid in an entry_spec if it begins with a hyphen. See the section "Notes on entry_specs".

-no_priv
Use the standard inquire_ calls, so that only entries that don't have their privacy flags set will be detected. This is the default.

-priv
Use the privileged calls to inquire_, so that entries whose privacy flags are set show up as existing. This requires access to the gate inquire_priv_.

-self
Used to specify the calling user in an entry_spec. See the section "Notes on entry_specs".

Notes on entry_specs:
Entry_specs are used to specify an Inquire database entry to be checked. Usually an entry_spec is just a userid, and the entry specified is the entry belonging to that user (by virtue of the specified userid matching the userid field of the entry). An entry_spec of -self can be used to specify the caller's entry. In order to specify a userid that begins with a hyphen the userid should

be preceded by the -name control argument.

Name: display_inquire_fields, diqf

The display_inquire_fields command prints or returns information about a Multics system user or users. This information is retrieved from the Inquire user database.

Syntax: diqf {entry_specs} {-control_args}

Syntax as an active function: [diqf {entry_specs} {-control_args}]

Arguments:

entry_specs
specify the users about whom information is to be returned (see the section "Notes on entry_specs). At least one entry must be specified, either by entry_spec arguments or by control arguments.

Control Arguments:

-all, -a
specifies that the information is returned about all the users in the database. This control argument is incompatible with entry_specs. This may also be used as the sole field_spec following -fields or -field_privacy_flags, and indicates that all the fields are desired (see the section "Notes on field_specs"). (1)

-entry_privacy_flag, -epvf
specifies that the value of the entry privacy flag is to be returned, as the value "true" or "false". This control argument is only valid if the entry is the caller's own entry or the -priv control argument is used, and is incompatible with the -last_name control argument.

-fields field_specs, -fl field_specs
specifies the fields of information to be returned about the specified user. If these arguments are not given, the contents of all the fields are returned, in the same order as the field names returned by the inquire_fields command. Each value returned is is

(1) Someone has suggested that this latter use of -all be replaced by two new control arguments, -all_fields and -all_field_privacy_flags, which would be mutually exclusive with -fields and -field_privacy_flags respectively, in order to make the syntax simpler.

requoted. See the section "Notes on field_specs" for their syntax.

-field_privacy_flags field_specs, -fpvf field_specs
specifies that the values of the privacy flags of the specified fields are to be returned, as the values "true" or "false". See the section "Notes on field_specs" for their syntax. This control argument is only valid if the entry is the caller's own entry or the -priv control argument is used, and is incompatible with the -last_name control argument.

-last_name NAMEs, -lnm NAMEs
specifies that information is to be returned about users whose last_name fields are one of the specified names. This control argument is incompatible with any of the control arguments that request privacy flag information.

-name STR, -nm STR
Used to specify a userid in an entry_spec or a field_name in a field_spec if it begins with a hyphen. See the sections "Notes on entry_specs" and "Notes on field_specs".

-no_priv
specifies that the normal inquire_ are to be used, so that only public data can be viewed. This is the default.

-priv
specifies that the privileged entries of inquire_ are to be used, thus bypassing much of the normal access control provided. Use of this control argument requires access to the gate inquire_priv_.

-self
Used to specify the calling user in an entry_spec. See the section "Notes on entry_specs".

-user entry_specs
specifies another entry to be used. This control argument is necessary when the entry_spec is following a field_spec or last name list. See the section "Notes on entry_specs".

Notes:
Multiple userids and last_names may be given and the requested information will be returned for each user.

When used as a command, display_inquire_fields prints the specified output for each user in turn, with an indication of which entry_spec or last name specified the entry, and with the actual data labeled. When used as an active function, only the requested data is returned. The order of this output is as follows: all entries specified by

entry_specs (duplicates are not suppressed) followed by all entries specified by last names. The order of the data returned for a particular entry is: field values, entry privacy flag, field privacy flags (any of these will be omitted if not requested). The fields are ordered in the order that they appeared on the command line, or in the order in which field names are returned by the inquire_fields command if -all was used as the field_spec.

Notes on entry_specs:
Entry_specs are used to specify Inquire database entries to be viewed. Usually an entry_spec is just a userid, and the entry specified is the entry belonging to that user (by virtue of the specified userid matching the userid field of the entry). An entry_spec of -self can be used to specify the caller's entry (this is recognized specially when performing operations that would normally imply -priv if an entry_spec were given). In order to specify a userid that begins with a hyphen the userid should be preceded by the -name control argument.

Notes on field_specs:
A field_spec is usually just a field name, as listed by the "inquire_fields" command. If the field name contains spaces or other special characters then it must be enclosed in doublequotes ("field name"), and if it begins with a hyphen then it must be preceded by the -name control argument to distinguish it from a control argument. If no other field_specs are specified, -all can be used to indicate that all existing fields are desired (they will be returned in the same order that the inquire_fields returns field names).

Name: inquire_fields, iqfl

The inquire_fields command prints or returns the names of the fields in the Inquire database. The order of the fields is the same as that of the output of the display_inquire_fields command when given no field name arguments.

Syntax: iqfl

Syntax as an active function: [iqfl]

Note: In the active function case, the field names are requoted before being returned.

Name: set_inquire_fields, siqf

The set_inquire_fields command allows a user to change parts of his entry in the Inquire database. If the entry did not previously exist it will be created (with the fields that are not set containing null strings and all privacy flags off).

Syntax: siqf {field_spec1 {-value} value1
{... field_specN {-value} valueN}} {-control_args}

Arguments:

field_specN
specifies the field of the Inquire database whose contents are to be set. See the section "Notes on field_specs".

valueN
is a string that is to be used as the new value for the field specified by field_specN. If the value contains blanks or other special characters it must be surrounded by quotes; if it begins with a hyphen it must be preceded by -value.

Control Arguments:

-entry_privacy_flag STR, -epvf STR
sets the privacy flag for the user's entry. STR can be either "on", "true", "off", or "false"; setting it on ("on" or "true") implies that other users may not view the entry.

-field_privacy_flags field_spec1 STR1 {... field_specN STRn},
-fpvf field_spec1 STR1 {... field_specN STRn}
sets the privacy flags for the specified fields of the user's entry. The STRn can be either "on", "true", "off", or "false"; setting a privacy flag on ("on" or "true") implies that other users may not determine the contents of the field in this entry (when they try they will get the null string).

-name STR, -nm STR
Used to specify a field_name in a field_spec if it begins with a hyphen. See the section "Notes on field_specs".

-value, -val
May precede the value argument of a field_spec/value pair. It is used if the value begins with a hyphen in order to distinguish it from a control argument.

Notes on field_specs:
A field_spec is usually just a field name, as listed by the "inquire_fields" command. If the field name contains spaces or other special characters then it must be enclosed in doublequotes ("field name"). If it begins with a hyphen then it must be preceded by the -name control argument to distinguish it from a control argument.

Name: fill_inquire_db

Syntax: fill_inquire_db {-control_args}

Function: Fills the Inquire database from the equivalent information in the User Registration File (URF). Inquire entries that already exist will not be modified. URF fields that have no corresponding fields in Inquire are not copied over; Inquire fields that have no corresponding URF fields will contain the null string. The field privacy flags for new entries are set off, and the entry privacy flags are set as selected by the user.

Control arguments:

-entry_privacy_flag STR, -epvf STR
specifies the entry privacy flag setting for Inquire entries created by this command. STR should be either "on" or "off". (Default: on)

Access required: The user must have read access to the URF and access to the gate inquire_priv_.

Name: inquire

Syntax: inquire {userid} {-control_args}

Function: The inquire subsystem allows the user to interactively update his entry and view other entries in the Inquire database.

Argument:
userid
The userid of the entry that is to be the initial current entry. Other entries may be viewed using subsystem requests. (DEFAULT - the user's own entry).

List of control arguments:

-abbrev, -ab
enables abbreviation expansion and editing of request lines.
-brief, -bf
shortens informative messages from inquire.
-enter_prompt STR, -enpt STR
sets the default prompt string to be used by the "enter" request. See the documentation of this request for more information on this prompt.
-long, -lg
prints full informative messages. (DEFAULT)
-no_abbrev, -nab
disables abbrev processing of request lines. (DEFAULT)
-no_priv
Specifies that inquire is to run in unprivileged mode; the standard inquire_ entrypoints are to be used when accessing the database. (DEFAULT)
-priv
Specifies that inquire should run in privileged mode; the privileged inquire_ entrypoints are to be used when accessing the database. Use of this control argument requires access to the gate inquire_priv_.
-profile PATH, -pf PATH
specifies the pathname of the profile to use for abbreviation expansion. The suffix ".profile" is added if necessary. This control argument implies -abbrev.
-prompt STRING
sets the request loop prompt to STRING. (default is "inquire:")
-quit
exit inquire after performing any operations specified in command line control arguments (such as -request).

-request STRING, -rq STRING
execute STRING as an inquire request line before entering the request loop.

Requests:

prints an message identifying the version of the inquire subsystem being used, the recursion level (if greater than one), the userid of the current entry, and whether he is in privileged mode.

delete {entry_specs} {-control_args}, dl {entry_specs} {-control_args}
deletes the specified entries from the Inquire database. The user will be queried before the entries are actually deleted. If an entry_spec specifies a nonexistent entry then the user will be notified, but the other entries will be deleted. In order to delete entries other than one's own the user must have access to the gate inquire_priv_. Control arguments are:

-all, -a
deletes all the entries in the database. This control argument is incompatible with any entry_spec arguments.
-force
deletes the entries without querying the user first.

enter {field_specs} {-control_args}, en {field_specs} {-control_args}
Prompts the user for new values for the specified fields (all the fields, if no field_specs are given). If the entry did not previously exist it will be created, with all the fields initially containing the null string and all privacy flags off.

The old value will be shown, and if a null response is entered the value is not changed. When all the values have been entered, the changes will be displayed and the user will be queried as to whether to make the changes. He may then answer "yes" to make the changes, "no" to get prompted for all of them again, or the name of a field that he wishes to reenter. Control arguments are:

-all, -a
specifies that all the fields are to entered. This control argument is incompatible with any field_spec arguments.
-documentation STR, -doc STR
sets the documentation flag and string for the IMMEDIATELY PRECEDING field_spec. If the STR is the null string, the documentation flag is set false, otherwise the documentation flag is set true and the documentation string is set to STR. STR can be an ioa_ control string of no arguments. See "Notes on

prompting" for more information on the use of this documentation string and documentation flag. (1)

-no_priv
Uses standard inquire_ calls, so the entry may only be modified if it is the user's own entry. (DEFAULT in unprivileged mode)

-priv
Uses privileged inquire_ calls, allowing modification of entries other than the user's own entry. Use of this control argument requires access to the gate inquire_priv_. (DEFAULT in privileged mode)

-prompt STR
specifies a prompt to be used when entering the value for the entry_specs which follow it (until another -prompt control argument). If STR is -default or -dft, the prompt reverts to the default. STR must be enclosed in quotes if it contains blanks or other special characters, and may be an ioa_ control string that takes the name of the field, the old value, a documentation flag and a documentation string as arguments. See "Notes on prompting" for more information. (DEFAULT: the argument to the -enter_prompt control argument to the subsystem invocation, or "^a: Old value:^/^a^2/^[(^a)^/^]New value:^/" if none was given.)

-user entry_spec
Specifies that the fields be changed in the specified entry. Unless the entry_spec is -self this control argument implies -priv and requires access to the gate inquire_priv_.

Notes: For each selected field, the old contents will be printed (by default) and the user will be asked to supply a new value. A value may take up multiple lines, and the response is ended with a line consisting only of a period (."). If the first line is blank (not even containing a period) then this is interpreted as a request to leave the field unchanged. Therefore, this request cannot be used to enter values with a leading blank line; use the set request for this. The final newline (the one before the period used to end the entry) is stripped from the value; in order to leave a trailing newline in the value leave a blank line BEFORE the line with the period. A null value may be entered by typing only a period on the first response line.

Notes on prompting: The -prompt control argument specifies a prompt to be used for all fields being entered with the particular invocation of the "enter" request. It may be an ioa_ control string, and receives the following arguments for each field that is being entered: the name of the field, the old value of that field

(1) If and when documentation strings are added to the database this will default to that string.

for the specified user (it will be the null string if the user previously had no inquire entry), a documentation flag, which indicates whether a documentation string was supplied for the field, and a documentation string (the null string if the documentation flag was false), which is meant to be per-field prompt information. The default prompt string is

```
"^a:  Old value:^/^a^2/^[(^a)^/^]New value:^/"
```

If the request line were

```
inquire: enter home_address -doc "Your street address"
home_phone
```

the dialogue might look like (user input is preceded by "!"):

```
home_address:  Old value:
474 Memorial Drive

(Your street address)
New value:
! 3 Ames St.
! .
home_phone:  Old value:
(617) 225-7594

New value:
! (617) 225-6232
! .
```

get {field_specs} {-control_args}, print {field_specs} {-control_args}, pr {field_specs} {-control_args}
displays the contents of the specified fields of selected entries. If no field_specs are specified, all are displayed. Control arguments may be:

-all, -a
: prints the contents of the specified fields in all entries in the database.

-last_name NAME, -lnm NAME
: Show the fields of the entries whose last_name field is NAME.

-no_priv
: Use the standard inquire_ entrypoints. (DEFAULT in unprivileged mode)

-priv
: Use the privileged inquire_ entrypoints. Use of this control argument requires access to the gate inquire_priv_. (DEFAULT in privileged mode.)

-user entry_spec
: Show the fields of the specified entry.

The -user and -last_name control arguments may be specified more than once and in combination; the fields will be displayed for all the specified entries. If neither is specified the current entry is used. If a particular entry is specified more than once it will be

duplicated in the output.

[get {field_specs} {-control_args}]
returns the contents of the specified fields of selected entries. Arguments are the same as for the get request. The values are requoted and returned in the same order as specified in the arguments (including duplicates), or, if no field_spec arguments are given, in the order that field names are returned from the list_fields active request.

get_privacy {field_specs} {-control_args},
gpriv {field_specs} {-control_args}
prints the values of the privacy flags of an entry. If no arguments are given, both the entry privacy flag and all the field privacy flags are printed, otherwise only those privacy flags specified by the arguments are printed. In order to print the value of the privacy flags for an entry other than one's own the user must have access to the gate inquire_priv_. Control arguments can be:

-all, -a
prints the requested privacy flag settings for all entries in the database. The userid of the entry is also printed. This control argument implies -priv and requires access to the gate inquire_priv_.
-entry, -et
prints the entry privacy flag, in addition to any field privacy flags explicitly specified by other arguments. (DEFAULT is to print the entry privacy flag only if no field_specs are given, along with all the field privacy flags.)
-no_entry, -net
does not print the entry privacy flag. (DEFAULT if field_specs specified.)
-no_priv
Uses standard inquire_ calls, so only the user's own entry's privacy flags may be viewed (the user's own entry is then the default for this request). (DEFAULT in unprivileged mode)
-priv
Uses privileged inquire_ calls, allowing viewing of privacy flags from other than the user's own entry. If this is specified, the default entry for this request is the current entry. Use of this control argument requires access to the gate inquire_priv_. (DEFAULT in privileged mode)
-user entry_spec
prints the privacy flags of the specified entry. If the entry_spec is not -self then -priv is implied by this argument. (DEFAULT is to print the privacy flags of the user's own entry, unless -priv is specified or the user is in privileged mode.)

[get_privacy {field_specs} {-control_args}],
[gpriv {field_specs} {-control_args}]
returns the settings of the specified privacy flags as truth values. Arguments are the same as for the get_privacy request. A value of "true" means that the corresponding privacy flag is on. If the entry privacy flag is being returned, it is the first value; the field privacy flag values are returned in the order that they were specified in the arguments; if no arguments are given that specify flags to be returned they are in the order that the fields are returned from the list_fields active request. The use of "-all" when this is used as an active request is not recommended, as no indication of the associated userid is returned along with the flag values; the "get" active request could be used for this, but there is no guarantee that the entries will be returned in the same order for the two requests. The recommended way to perform this operation is to use command line iteration over the userids returned by the active request "[get userid -all]", i.e.
do "if [gpriv &1 -et] -then ""e string &1"" " ([get userid -all])
would print the userids of all private entries.

list_fields, lfl
Prints the names of all the fields in the Inquire database.

[list_fields], [lfl]
returns the names of all the fields in the Inquire database.

quit, q
exits the inquire subsystem.

set field_spec1 {-value} value1 {... field_specN {-value} valueN}
{-control_args}
sets new values for the specified fields. If the entry did not previously exist it will be created, all the unspecified fields set to the null string and all privacy flags off.

At least one field_spec/value pair must be specified. The values may be ioa_ control strings of no arguments in order to insert carriage control operations (i.e. multiple-line values); they may also be -query, in which case the user will be prompted for the new value. The value must be enclosed in quotes if it contains spaces or other special characters, and must be preceded by -value if it begins with a hyphen. Control arguments are:

-no_priv
Uses standard inquire_ calls, so the entry may only be modified if it is the user's own entry. (DEFAULT in unprivileged mode)
-priv
Uses privileged inquire_ calls, allowing modification of entries other than the user's own entry. Use of this control argument

requires access to the gate inquire_priv_. (DEFAULT in privileged mode)

-user entry_spec
Specifies that the fields be changed in the specified entry. Unless the entry_spec is -self this control argument implies -priv and requires access to the gate inquire_priv_. (DEFAULT: if -priv was specified or the user is in privileged mode this defaults to the current entry; otherwise it defaults to the user's own entry.)

-value STR, -val STR
optionally precedes a value in a field_spec/value pair. It should be used when the STR begins with a hyphen in order to distinguish it from a control argument.

Notes on -query:
The response for the -query argument may be entered on multiple lines, and is ended with a line consisting only of a period (".").
This value is not processed for ioa_ control codes.

set_privacy {field_spec1 STR1 ... field_specN STRn} {-control_args},
spriv {field_spec1 STR1 ... field_specN STRn} {-control_args}
sets the privacy flags of an entry. If the entry did not previously exist it will be created, with all the fields set to the null string and any unspecified privacy flags off. The STR arguments can be "on" or "true" to make the specified field unreadable, or "off" or "false" to make it publicly readable. Control arguments are:

-entry STR
sets the entry privacy flag. This must be given if no field_specs are specified. STR can be "on" or "true", making the entry private, or "off" or "false", in which case it is publicly readable.

-no_priv
Uses standard inquire_ calls, so the entry may only be modified if it is the user's own entry. (DEFAULT in unprivileged mode)

-priv
Uses privileged inquire_ calls, allowing modification of entries other than the user's own entry. Use of this control argument requires access to the gate inquire_priv_. (DEFAULT in privileged mode)

-user entry_spec
Privacy flags should be changed in the specified entry. Unless the entry_spec is -self this control argument implies -priv and requires access to the gate inquire_priv_. (DEFAULT: in unprivileged mode, the user's own entry; in privileged mode (or when -priv is given), the current entry)

user {entry_spec {-control_args}}
If the entry_spec is given, sets the current entry to that entry;

otherwise, the userid of the current entry is printed. Control arguments are:

-no_priv
Specifies that normal inquire_ calls be used to find the specified entry. (DEFAULT in unprivileged mode)
-priv
Specifies that privileged inquire_ calls be used to find the specified entry, so that the current entry can be set to an entry that has its privacy flag set. Use of this control argument requires access to the gate inquire_priv_. (DEFAULT in privileged mode)

Notes: If the entry is not found, the current entry remains unchanged.

In addition to the above inquire requests, the following standard subsystem requests are also provided. Documentation for these requests can be found in the documentation of the ssu_ subroutine in the MPM Subsystem Writer's Guide (AK92) and MTB-540, which describes the ssu_ subsystem utilities.

.. COMMAND_LINE	escape a command line to Multics.
?	Print a list of available requests.
help	Obtain detailed information on inquire.
list_help, lh	List topics for which help is available.
list_requests, lr	List brief information on inquire requests.
abbrev, ab	Turn abbreviation processing on or off.
answer	Supply an answer to a question asked by a request.
do	Expand request string before passing to inquire.
execute, e	Execute a Multics command line.
if	Conditionally execute a request.
ready, rdy	Print a ready message.
ready_off, rdf	Turn ready messages off.
ready_on, rdn	Turn ready messages on.
subsystem_name	Print/return the name of the subsystem
subsystem_version	Print/return the current version number of inquire

Notes on field_specs:
Field_specs are used to specify fields in an Inquire entry to be operated on by several subsystem requests. A field_spec is usually just a field name, as listed by the "list_fields" request. If the

field name contains spaces or other special characters then it must be enclosed in doublequotes ("field name"). If it begins with a hyphen then it must be preceded by the -field control argument to distinguish it from a control argument.

Notes on entry_specs:
Entry_specs are used to specify an Inquire database entry to be operated on by an inquire request. If an entry_spec is not supplied to a request, it generally defaults to the "current entry," which is set by specifying an entry_spec on the inquire command line or with the "user" request. Usually an entry_spec is just a userid, and the entry specified is the entry belonging to that user (by virtue of the specified userid matching the userid field of the entry). An entry_spec of -self can be used to specify the caller's entry, and -default specifies the current entry. The -self entry_spec should be used if a request assumes -priv when an entry_spec is supplied (i.e. set_priv) in order to use the standard entrypoints and avoid errors due to insufficient access. In order to specify a userid that begins with a hyphen the userid should be preceded by the -name control argument.