To: MTB Distribution

From: Jim Gray

Date: March 26, 1981

Subject: Changes to the MRDS dmd_ Subroutine Interface.

Send comments by:

Multics mail on System M to JGray.Multics

Telephone to HVN 341-7463 or 602-249-7463

Continuum meeting mrdsdev, link to transaction 330.

1.0 INTRODUCTION

This MTB describes changes that are to be made in the MRDS dmd_ subroutine interface. The purpose of this interface is to provide a means of obtaining information about a MRDS database model. It is currently documented in Section 6, Subsystem Writers Guide portion of [1].

There are two major reasons for the changes being made. The first is the new MRDS security work, that is to provide attribute level access control, as outlined in [2], and [3]. The second has to do with TR's, such as 7072, and 7163, against the existing interface. These reasons are detailed in the next two sections.

2.0 PROBLEMS RAISED BY SECURITY

The terms used here, are those introduced in [2]. That document addresses the problems of data access security, as enforced through using submodel views of the data. Little is said about security for model information, which is not to be known in general by users of a restricted submodel view.

Currently, any user with "r" access to the model can use the dmd_ interface to find out relation and attribute specifications, regardless of what view his submodel might present. For a database that has been secured, the non-DBA user must not be allowed to see things outside his submodel view of the data, and the model.

Also, the passing of a pointer to the database model segment, as a means of communication between the various dmd_ entries, is a source of possible security breech. The user could avoid calling the open routine, obtain his own pointer, and access model information that the open routine might have restricted him from seeing.

There are cases where the model information must not be restricted. These include the user being a DBA, or internal MRDS code acting on behalf of a non-DBA, where the view has already been restricted.

3.0 PROBLEMS WITH THE EXISTING INTERFACE

The dmd_$get_attributes entry has several problems. The first being, if a -decode_dcl option was used in the data model source input to create_mrds_db, then the descriptor returned by this interface is not the users view of the data, as it should be, but the database view instead.

Second, the bit_offset and bit_length information returned present the internal MRDS tuple structure to the user. Thus offsets are returned as "substr" indexes rather than true offsets for fixed length attributes, and for varying attributes, an array index, not an offset is returned. The user should "see" the tuple in an expected, and helpful format, not the internal structure that MRDS uses, which can be confusing.

The dmd_$open_dm entry has an open mode parameter, whose use became archaic after the last release, as the open routine can now no longer be used to create databases. That function was taken over by the self contained dmd_$create_db routine.

None of the structures passed back in the dmd_ interface have a version number, making it impossible to extend or modify this interface, without breaking existing users.

4.0 OVERCOMING SECURITY PROBLEMS

The existing dmd_ interface will be made available to only DBA's, once the database is secured. If there is need for internal MRDS to use this interface, this restriction will not apply.

This will be done by having two entries in the module implementing each current entry. Thus for dmd_$open_dm, which references the module mrds_dm_open$open_dm, non-DBA's are refused once the database is secured. Other internal MRDS code will only call the entry mrds_dm_open$mrds_dm_open, which will have no restriction, so that commands such as create_mrds_dm_include, that make use of it, even though a non-DBA is accessing the database through a submodel view, will still work properly.

The documentation for the dmd_ subroutine entries will be changed to show that this interface is only available to a DBA once the database has been secured.

5.0 EXTENDING THE INTERFACE

To solve existing problems with this non-extensible interface, the dmd_ interface documentation will be moved to a new section of the MRDS manual for "obsolete interfaces". It will be replaced by an extensible interface that solves the current outstanding problems, called mmi_ (MRDS model interface).

The new mmi_ interface will not give bit_offset information about the tuple to avoid showing the internal tuple structure used by MRDS.

The descriptor returned for attributes will represent the users data view, in case a -decode_dcl option is present.

Versioned structures will be used, with an input structure number that specifies which version of the structure the user wishes to use. This will allow new structure versions to be developed, without breaking existing applications. The same dmd_ method of allocation of a structure into an area will be used, even if the structure is fixed length, in order to provide for extensiblity of the structure, possibly to a varying length.

The passing of a pointer for interface communication will be discarded. Instead, a user supplied opening name of arbitrary length and character make up, will be used to associate an opening of the model, with the relavent information. The open name manager supplied for this will also be usable by the new msmi_ interface, to be documented in [4].

Two new entries will be added to the functionality of mmi_ that otherwise duplicates that already available from the dmd_ interface. One will be for determining the secured state of the database, and the other is for determining if the caller is a DBA.

The mmi_ interface will not be used internally in MRDS. This will avoid user problems that might arise because of changing internal MRDS needs.

Once the database is secured, the mmi_ model information entries get_model_info, get_model_attributes, get_model_relations, and open_model will require the caller to be a DBA.

6.0 MMI_ DOCUMENTATION

NAME: mmi_

This subroutine primarily provides a means of retrieving information about a database model (Mrds_Model_Interface_). There is also an entry to create a database in the same manner as the create_mrds_db command. See the msmi_ subroutine interface for submodel information.

ENTRY: mmi_$close_model

This entry closes a given opening of the database model.

USAGE

```
declare mmi_$close_model entry (char(*), fixed bin (35)) ;

call mmi_$close_model (opening_name, error_code) ;
```

WHERE:

1. opening_name (Input) (char(*))
   is the name given in the call to mmi_$open_model, for the opening of the model that is to be closed.

2. error_code (Output) (fixed bin (35))
   is a standard status code. If the name given does not refer to a current model opening, the code mrds_error_$open_name_not_known will be returned.

ENTRY: mmi_$create_db

This entry provides a go/no-go subroutine interface to create_mrds_db.

USAGE

```
     declare mmi_$create_db entry options (variable);

     call mmi_$create_db ("source_path", {"db_path",} {"-list",}
          {"-secure",} {"-temp_dir", "temp_dir_path",} {"-force"}
          code);
```

where the arguments are the same character string arguments as given at command level to the create_mrds_db command except that code must be declared fixed bin(35). The same option and features are available. However, the error code of the first error encountered is returned since it is a go/no-go interface.

NOTES

Since create_mrds_db was written for command level, some of its error codes do not provide much detail, therefore a listing should be requested to provide full information.

If the -temp_dir {path} is given, path should be a separate character string argument from "-temp_dir".

If character variables, rather than constants, are used in the call to mmi_$create_db, then trailing blanks should be suppressed (e.g., with the PL/I built-in "rtrim", described in the PL/I Language Specification).

ENTRY: mmi_$get_authorization

This entry returns the user class of the caller, for a given database.

USAGE

```
     declare mmi_$get_authorization entry
          (char(*), ptr, fixed bin, ptr, fixed bin(35)) ;

     call mmi_$get_authorization (database_path, area_ptr,
          structure_version, mrds_authorization_ptr,
          error_code) ;
```

WHERE:

1. database_path (Input) (char(*))
   is the relative or absolute pathname of the database, with or without the ".db" suffix. This path must refer to a version 4 database.

2. area_ptr (Input) (pointer)
   is a pointer to a freeing area supplied by the caller, in which the mrds_authorization structure is to be allocated.

3. structure_version (Input) (fixed bin)
   is the desired structure version the user wishes to have returned.

4. mrds_authorization_ptr (Output) (pointer)
   is a pointer to the allocated structure. This structure is described in the Notes below.

5. error_code (Output) (fixed bin (35))
   is a standard status code. It may be one of the following:

   error_table_$badcall if the area_ptr was null

   error_table_$area_too_small if the supplied area could not contain the mrds_authorization structure

   mrds_error_$not_freeing_area if the supplied area does not have the attribute "freeing".

   error_table_$unimplemented_version if the given structure version is unknown

   mrds_error_$version_not_supported if the database path does not refer to a version 4 MRDS database mrds_error_$no_database if the given path does not refer to a MRDS database.

NOTES

The user class information for the specified database is returned in the following structure (see Appendix F for the include file mrds_authorization.incl.pl1) :

```
     declare 1 mrds_authorization aligned
              based (mrds_authorization_ptr),
             2 version fixed bin,
             2 administrator bit (1) unal,
             2 normal_user bit (1) unal,
             2 mbz bit (34) unal ;
```

WHERE:

1. version
   is the version number of this structure

2. administrator
   is "1"b, if the caller is a DBA

3. normal_user
   is "1" if the caller is a non-DBA. Note that a DBA is always also a normal user.

4. mbz
   is reserved for future use

Currently, the only available structure version is 1.

The user must have sufficient access to get the effective access mode on the database directory.

ENTRY: mmi_$get_model_attributes

This entry returns attribute information for a particular relation in the database model.

USAGE

```
     declare mmi_$get_model_attributes entry
          (char(*), char(*), ptr, fixed bin,
          ptr, fixed bin(35)) ;

     call mmi_$get_model_attributes (opening_name, relation_name,
          area_ptr, structure_version,
          mrds_db_model_rel_attrs_ptr, error_code) ;
```

WHERE:

1. opening_name (Input) (char(*))
   is the name used in the call to mmi_$open_model

2. relation_name (Input) (char(*))
   is the name of the relation for which the attribute information is desired

3. area_ptr (Input) (pointer)
   is a pointer to a user supplied freeing area in which the attribute information will be allocated.

4. structure_version (Input) (fixed bin)
   is the desired version of the attribute information structure to be allocated.

5. mrds_db_model_rel_attrs_ptr (Output) (pointer)
   is a pointer to the allocated attribute information structure described in the Notes below.

6. error_code (Output) (fixed bin (35))
   is the standard status code. It may be one of the following:

   error_table_$badcall if the area_ptr was null

   error_table_$area_too_small if the supplied area could not hold the attribute information structure

   mrds_error_$not_freeing_area if the supplied area does not have the attribute "freeing".

   error_table_$unimplemented_version if the structure version given was unknown mrds_error_$bad_relation_name if the relation name given is not in the model definition

mrds_error_$open_name_not_known if the name given does not refer to a current model opening

NOTES

The attribute information is returned in the following structure (see Appendix F for the include file mrds_db_model_rel_attrs.incl.pl1) :

```
     declare 1 mrds_db_model_rel_attrs aligned
              based (mrds_db_model_rel_attrs_ptr),
             2 version fixed bin,
             2 attribute_count fixed bin,
             2 mbz1 bit (36) unal,
             2 attribute (0
              refer (mrds_db_model_rel_attrs.attribute_count)),
               3 name char (32),
               3 domain char (32),
               3 user_data_type bit (36),
               3 indexed bit (1) unal,
               3 mbz2 bit (35) unal ;
```

WHERE:

1. version
   is the version number of this structure

2. attribute_count
   is the number of attributes in this relation

3. mbz1
   reserved for future use

4. name
   is the name of this attribute

5. domain_name
   is the name of the underlying domain for this attribute

6. user_data_type
   is a standard Multics descriptor for the users view of the data in this domain. It will differ from the database data type if the -decode_dcl option was used for this domain.

7. indexed
   is "1"b, if the attribute is the total key, a key head, or secondary index in the relation

8. mbz2
   reserved for future use.

Currently the only structure version available is 1.

If the database is secured, this interface is only usable by a DBA. If the database is not secured, the user must have "r" access to the model segment for the relation involved.

ENTRY: mmi_$get_model_info

This entry returns information about the database model creation.

USAGE

```
     declare mmi_$get_model_info entry
          (char(*), ptr, fixed bin, ptr, fixed bin(35)) ;

     call mmi_$get_model_info (opening_name, area_ptr,
          structure_version, mrds_db_model_info_ptr,
          error_code) ;
```

WHERE:

1. opening_name (Input) (char(*))
   is the name used in the call to mmi_$open_model

2. area_ptr (Input) (pointer)
   is a pointer to a user supplied freeing area in which the model information will be allocated

3. structure_version (Input) (fixed bin)
   is the desired structure version of the model information

4. mrds_db_model_info_ptr (Output) (pointer)
   the pointer to the allocated model information structure as described in the Notes below.

5. error_code (Output) (fixed bin (35))
   is the standard status code. it may be one of the following:

   error_table_$badcall if the area_ptr was null

   error_table_$area_too_small if the area could not hold the model information structure

   mrds_error_$not_freeing_area if the supplied area does not have the attribute "freeing".

   error_table_$unimplemented_version if the supplied structure version is unknown

   mrds_error_$open_name_not_known if the opening_name does not refer to a current model opening

NOTES

The model information is returned in the following structure (see Appendix F for the include file mrds_db_model_info.incl.pl1):

```
     declare 1 mrds_db_model_info aligned
                based (mrds_db_model_info_ptr),
               2 version fixed bin,
               2 model_version fixed bin,
               2 creator_id char (32),
               2 creation_time fixed bin (71),
               2 mbz bit (36) unal ;
```

WHERE:

1. version
   is the version number of this structure

2. model_version
   is the database version. The latest version is 4.

3. creator_id
   is in the form person.project.tag as returned from get_group_id_, for the creator of the database

4. creation_time
   is the time the database was created, in a form acceptable to date_time_

5. mbz
   reserved for future use

Currently, the only structure version available is 1.

If the database is secured, this interface is only usable by a DBA. If the database is not secured, the user must have "r" access to the db_model segment under the database directory.

ENTRY: mmi_$get_model_relations

This entry returns information about all the relations in the given model opening.

USAGE

```
     declare mmi_$get_model_relations entry
          (char(*), ptr, fixed bin, ptr, fixed bin(35)) ;

     call mmi_$get_model_relations (opening_name, area_ptr,
          structure_version, mrds_db_model_relations_ptr,
          error_code) ;
```

WHERE:

1. opening_name (Input) (char(*))
   is the name used in the call to mmi_$open_model

2. area_ptr (Input) (pointer)
   is a pointer to a user supplied freeing area in which the relation information will be allocated

3. structure_version (Input) (fixed bin)
   is the desired structure version of the relation information

4. mrds_db_model_relations_ptr (Output) (pointer)
   is the pointer to the allocated structure of relation information in the form described in Notes below.

5. error_code (Output) (fixed bin (35))
   is the standard status code. It may be one of the following:

   error_table_$badcall if the area_ptr was null

   error_table_$area_too_small if the area could not hold the relation information

   mrds_error_$not_freeing_area if the supplied area does not have the attribute "freeing".

   error_table_$unimplemented_version if the given structure version is unknown

   mrds_error_$open_name_not_known if the opening_name does not refer to a current model opening.

NOTES

The relation information is returned in the following structure (see Appendix F for the include file mrds_db_model_relations.incl.pl1) :

```
     declare 1 mrds_db_model_relations aligned
             based (mrds_db_model_relations_ptr),
            2 version,
            2 relation_count fixed bin,
            2 mbz1 bit (36) unal,
            2 relation (0
             refer (mrds_db_model_relations.relation_count)),
              3 name char (32),
              3 mbz2 bit (36) unal ;
```

WHERE:

1. version
   is the version number of this structure

2. relation_count
   is the number of relations defined in the model

3. mbz1
   is reserved for future use

4. name
   is the name of this relation

5. mbz2
   is reserved for future use

Currently, the only structure version available is 1.

If the database is secured, this interface is usable only by a DBA. If the database is not secured, the user must have "r" access to the db_model segment under the database directory.

ENTRY: mmi_$get_secured_state

This entry the secured state of the database for the given opening.

USAGE

```
     declare mmi_$get_secured_state entry
          (char(*), ptr, fixed bin, ptr, fixed bin(35)) ;

     call mmi_$get_secured_state (database_path, area_ptr,
          structure_version, database_state_ptr,
          error_code) ;
```

WHERE:

1. database_path (Input) (char(*))
   is the relative or absolute pathname of the database whose secured state is desired. It must refer to a version 4 database. The suffix need not be present.

2. area_ptr (Input) (pointer)
   is a pointer to a user supplied freeing area in which the database state information will be allocated

3. structure_version (Input) (fixed bin)
   is the desired version of the structure containing database state information

4. database_state_ptr (Output) (pointer)
   the pointer to the allocated database state information as contained in the structure described in the Notes below.

5. error_code (Output) (fixed bin (35))
   is the standard status code. It may be one of the following:

   error_table_$badcall if the area_ptr was null

   error_table_$area_too_small if the supplied area could not hold the database state information

   mrds_error_$not_freeing_area if the supplied area does not have the attribute "freeing".

   error_table_$unimplemented_version if the supplied structure version is unknown

   mrds_error_$version_not_supported if the path given is to a database whose version is less than 4 mrds_error_$no_database if the given path does not refer to a MRDS database.

mrds_error_$no_model_access if the user does not "r" access to the database db_model segment

NOTES

The database state information is returned in the following structure (see Appendix F for the include file mrds_database_state.incl.pl1) :

```
     declare 1 database_state aligned
                based (database_state_ptr),
               2 version fixed bin,
               2 unsecured bit (1) unal,
               2 secured bit (1) unal,
               2 mbz bit (34) unal ;
```

WHERE:

1. version
   is the version number of this structure

2. unsecured
   is "1"b, if the database is not currently secured

3. secured
   is "1"b if the database is currently secured

4. mbz
   reserved for future use

Currently, the only structure version available is 1.

The user must have at least "r" access to the db_model segment under the database directory.

ENTRY: mmi_$open_model

This entry opens the database model for retrieving model information about relations, attributes, or creation info. There may be multiple openings of the same database model, or different database models.

USAGE

```
     declare mmi_$open_model entry
          (char(*), char(*), fixed bin (35)) ;

     call mmi_$open_model (database_path, opening_name,
          error_code) ;
```

WHERE:

1. database_path (Input) (char(*))
   is the relative or absolute pathname of the database, whose data model is to be opened. Version 4 databases need not have the ".db" suffix supplied.

2. opening_name (Input) (char(*))
   a user supplied name, to be used in other mmi_ calls referencing this opening when obtaining model information.

3. error_code (Output) (fixed bin (35))
   is the standard status code. It may be one of the following:

   mrds_error_$open_name_already_known if the opening_name supplied was not unique, within PL/I comparison rules, of other opening names already used in the users process

   mrds_error_$too_many_open_names if the combined lengths and number of opening_names used in the users process exceeded the storage capability of the open name manager

   mrds_error_$no_database if no database exists at the given pathname

   mrds_error_$no_model_access if the user does not have "r" access to the database model segment.

   error_table_$insufficient_access if the database has been secured and the user is not a DBA

NOTES

The opening_name may be any number of ascii characters. Current capability is for more than 1000 opening_names of reasonable length. Opening_names must be unique within PL/I comparison rules within the users process. (the entry unique_chars_, described in MPM Subroutines, can be used to generate unique names)

If the database is secured, this interface is only usable by a DBA. If the database is not secured, the user must have at least "r" access to the db_model segment under the database directory.

7.0 REFERENCES

[1] Multics Relational Data Store Reference Manual, Order number AW53-03

[2] The New MRDS Security Approach, MTB-501

[3] Effects of Security on the MRDS Interface, MTB-502

[4] Changes in the MRDS Submodel Interface, MTB-496

[5] Changes to the MRDS Command Interface, MTB-503

[6] Changes to the MRDS dsl_ Subroutine Interface, MTB-504

[7] Extensions to the create_mrds_dsm and display_mrds_dsm Commands for MRDS Security, MTB-506