To: MTB Distribution

From: Jim Gray

Date: March 24, 1981

Subject: Changes to the MRDS dsl_ 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 331.

1.0 INTRODUCTION

This MTB describes changes to the data manipulation subroutine interface to MRDS. This interface is currently documented in Section 4, Data Sublanguage Subroutines, of [1].

There are three primary motivations for these changes. The first is the addition of a new attribute level access control mechanism to MRDS. The terms used in this document, when referring to the new security scheme, can be found in the overview documents [2], and [3]. The second is the change in concurrency control modes as detailed in MCR 4812, and as needed for the new security approach. The third is for fixing problems with the existing interface, such as reported in TR's 7074, 7780, 8133, 8424, 8990, 8991, and 9215.

2.0 PROBLEMS OUTLINED IN TR'S

The entry dsl_$list_dbs currently does not work for version 3 and earlier databases. The entry does not provide for an error code. There is need to add opening mode information to the returned structure, but the structure has no version number.

The entry dsl_$get_temp_dir is not properly designed. It can not return the temporary storage directory for a particular opening, only that for the next opening to be made. It needs to have the same functionality as display_mrds_temp_dir.

The dsl_$get_db_version entry handles submodels in a clumsy way, that requires the caller to do several string operations on the returned path, to determine if the input path refers to a submodel.

The dsl_$open entry has bad performance because even relations whose data is not referenced during the life of the opening have their vfiles attached and opened several times.

There is no interface for determining the number of tuples in either permanent or temporary relations. There is also no way to determine the number of tuples specified by a given selection expression.

There is no simple subroutine means by which to display the scope the user has set on relations in a shared opening of the database.

If a user wishes to build selection expressions with string operations on varying length character strings, and substitute these into calls to dsl_$retrieve, dsl_$modify, dsl_$delete, and dsl_$define_temp_rel, he is prevented from doing so. This is because currently only an argument declared char(N) will be acceptable, even though these interfaces are declared options (variable), and could determine the data type of their arguments.

3.0 PROBLEMS RELATED TO THE NEW SECURITY APPROACH

The new approach to security provides for attribute level access control by forcing all openings to be through submodels residing under the database once the database is secured. To implement this, the behavior of dsl_$open will have to be changed.

Further, the MR8 dsl_$open detects access violations at open time based on what the vfile attachment detects. This happens for all relations in the database, even if they will not be referenced during the life of the opening.

There is currently no interface capable of displaying access information since dsmd_$validate_rel was removed with the last release of MRDS. This interface also provided scope information, and lists of relation and attribute descriptions.

The read-update scope modes, changed back to read - store - modify - delete by MCR 4812, do not have a common ground of meaning with the new security approach access modes. This is because the scope modes work at a relation level, with no consideration for future attribute level granularity. The new attribute level access modes are read_attr and modify_attr, with the relation level modes being append_tuple and delete_tuple.

4.0 CHANGES TO DSL_ ENTRIES

The entries dsl_$list_dbs and dsl_$get_db_version will be placed in a new section of the MRDS manual for "obsolete interfaces", and they will be replaced by the new entries dsl_$list_openings and dsl_$get_path_info that will be extensible interfaces using version-ed structures containing the additional needed information.

The entry dsl_$get_temp_dir will be augmented by a new entry dsl_$get_opening_temp_dir that will be able to return the temporary storage directory for a given opening.

The new entry dsl_$get_scope will provide a convenient means of retrieving scope that the user has set in a particular opening. See [4] for interfaces that provided similar functions at command level.

The data manipulation entries that accept a selection expression will be changed to work with either a char(N) or char(N) varying argument for the selection expression.

A new entry dsl_$get_population will provide for determining the current tuple count of permanent and temporary relations, and by means of defining a temporary relation, the number of tuples selected by a particular selection expression.

The attachment and opening of vfiles containing relation data will be moved from dsl_$open time, until after the time that scope is set on the relation (scope must always be set before data can be accessed, either explicitly by the user, or implicity for the user by MRDS).

The entry dsl_$set_scope (whether called implicitly or explicitly) will now detect Multics acl violations on the data, and MRDS access violations. This will prevent users from locking portions of the database that they do not have access to.

The new entries dsl_$get_relation_list and dsl_$get_attribute_list will be added to provide for obtaining access information and to replace the lost functionality of dsmd_$validate_rel.

The scope interfaces will have the current store - read - modify - delete names changed to read_attr - append_tuple - modify_attr - delete_tuple in the display routines and in the documentation. This will allow the user to clearly see the permissable operations that his MRDS access allows.

4.1 SUMMARY OF DSL_ CHANGES

OBSOLETED ENTRIES

dsl_$get_db_version	(replaced by dsl_$get_path_info)
dsl_$list_dbs	(replaced by dsl_$list_openings)

NEW ENTRIES

dsl_$get_attribute_list	(gets access info)
dsl_$get_opening_temp_dir	(augments dsl_$get_temp_dir)
dsl_$get_path_info	(replaces dsl_$get_db_version)
dsl_$get_population	(gets tuple counts)
dsl_$get_relation_list	(gets access info)
dsl_$get_scope	(retrieves scope settings)
dsl_$list_openings	(replaces dsl_$list_dbs)

CHANGED ENTRIES

dsl_$define_temp_rel	(char varying select expr)
dsl_$delete	(char varying select expr)
dsl_$dl_scope	(new scope modes)
dsl_$get_temp_dir	(only documentation)
dsl_$modify	(char varying select expr)
dsl_$open	(effect of secured db)
dsl_$retrieve	(char varying select expr)
dsl_$set_scope	(new scope modes)
dsl_$set_scope_all	(new scope modes)

UNCHANGED ENTRIES

dsl_$close
dsl_$close_all
dsl_$declare
dsl_$delete_scope_all
dsl_$set_temp_dir
dsl_$store

5.0 NEW DSL_ ENTRIES

5.1 GET_ATTRIBUTE_LIST

ENTRY: dsl_$get_attribute_list

This entry returns information on the attributes in the view of the given relation provided by the users opening.

USAGE

```
     declare dsl_$get_attribute_list entry (fixed bin (35),
          char(*), ptr, fixed bin, ptr, fixed bin(35)) ;

     call dsl_$get_attribute_list (db_index, relation_name,
          area_ptr, structure_version, mrds_attribute_list_ptr,
          error_code) ;
```

WHERE:

1. db_index (Input) (fixed bin(35))
   is the integer returned by dsl_$open for the opening the user wishes to reference

2. relation_name (Input) (char(*))
   is the name of the relation in the users view, 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 is to be allocated.

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

5. mrds_attribute_list_ptr (Output) (pointer)
   is a pointer to the attribute information returned in a structure as 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

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

error_table_$unimplemented_version if the structure_version supplied is unknown

mrds_error_$unknown_relation_name if the given relation name is not known in this openings view of the database.

NOTES

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

```
     declare 1 mrds_attribute_list aligned
              based (mrds_attribute_list_ptr),
             2 version fixed bin,
             2 access_info_version fixed bin,
             2 num_attrs_in_view fixed bin,
             2 submodel_view bit (1) unal,
             2 mbz1 bit (35) unal,
             2 attribute (0
              refer (mrds_attribute_list.num_attrs_in_view)),
               3 model_name char (32),
               3 submodel_name char (64),
               3 domain_name char (32),
               3 user_data_type bit (36),
               3 system_acl char (8) varying,
               3 mrds_access char (8) varying,
               3 effective_access char (8) varying,
               3 indexed bit (1) unal,
               3 mbz2 bit (35) unal ;
```

WHERE

1. version
   is the version number of this structure.

2. access_info_version
   is the version of the mrds access modes returned in the attriubte information. Version 3 access_info_version refers to version 3 databases with r-s-m-d relation access modes. Version 4 refers version 4 databases without attribute level security, using r-e-w system acls. Version 5 refers to version 4 databases with attribute level security using read_attr (r) and modify_attr (m) attribute access modes.

3. num_attrs_in_view
   is the number of attributes in this openings view of the given relation.

4. submodel_view

is "1"b, if this opening referred to with db_index was through a submodel.

5. mbz1
   is reserved for future use.

6. model_name
   is the name of this attribute in the database model. If the database is secured, and the caller is not a DBA, then this field will be blanks.

7. submodel_name
   is the name of the attribute in the submodel view, if the opening referred to by db_index was through a submodel, otherwise it is the same as the model name.

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

9. user_data_type
   is the standard Multics descriptor for the data type of this domain. It represents the users view if a -decode_dcl option was used for the domain.

10. system_acl
    is the Multics acl on this attribute, from the modes r-e-w

11. mrds_access
    is the mrds access mode for this attribute, see the access_info_version description for possible values, for various of mrds access control

12. effective_access
    is the result of applying both system acl's and mrds access to this attribute, using mrds access values for the effect

13. indexed
    is "1"b, if this attribute is the total key, the key head attribute, or a secondarily indexed attribute.

14. mbz2
    is reserved for future use

Currently, the only structure version available is 1.

5.2 GET_OPENING_TEMP_DIR

ENTRY: dsl_$get_opening_temp_dir

This entry returns the pathname of the directory that is being used for temporary storage for a particular database opening.

USAGE

```
     declare dsl_$get_opening_temp_dir entry
          (fixed bin(35), fixed bin(35)) returns(char(168)) ;

     path = dsl_$get_opening_temp_dir(db_index, error_code) ;
```

WHERE:

1. db_index (Input) (fixed bin(35))
   is the integer returned by a call to dsl_$open, and refers to the opening whose temporary storage directory is desired

2. error_code (Output) (fixed bin(35))
   is the standard status code. It will be mrds_error_$invalid_db_index, if the supplied db_index does not refer to a currently open database in the users process.

3. path (Output) (char(168)
   is the absolute pathname of the directory being used for temporary storage for the opening specified.

NOTES

See dsl_$get_temp_dir for an entry that will return the directory pathname that will be used in the next call to open. Also see dsl_$set_temp_dir and the commands display_mrds_temp_dir and set_mrds_temp_dir.

5.3 GET_PATH_INFO

ENTRY: dsl_$get_path_info

This entry returns information about a supplied pathname. It indicates whether or not the path refers to a MRDS database model or submodel, and if so, what the version is, and details about it's creation.

USAGE

```
     declare dsl_$get_path_info entry(char(*), ptr,
          fixed bin, ptr, fixed bin(35)) ;

     call dsl_$get_path_info(in_path, area_ptr,
          structure_version, mrds_path_info_ptr,
          error_code) ;
```

WHERE:

1. in_path (Input) (char(*))
   is the relative or absolute pathname about which the user desires information. If it refers to a MRDS database model or submodel, it does not need a suffix, unless ambiguity would result. A model will be found before the submodel, if they both have the same name, less suffix, in the same directory.

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

3. structure_version (Input) (fixed bin)
   is the desired version of the path information structure to be returned

4. mrds_path_info_ptr (Output) (pointer)
   is the pointer to the path information structure that is returned, which is 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 path 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_$no_model_submodel if the path does not refer to a MRDS database model or submodel

NOTES

The path information is returned in the following structure (see Appendix F for the include file mrds_path_info.incl.pl1)

```
declare 1 mrds_path_info aligned
         based (mrds_path_info_ptr),
        2 version fixed bin,
        2 absolute_path char (168),
        2 type,
          3 not_mrds bit (1) unal,
          3 model bit (1) unal,
          3 submodel bit (1) unal,
          3 mbz1 bit (33) unal,
        2 mrds_version fixed bin,
        2 creator_id char (32),
        2 creation_time fixed bin (71),
        2 mbz2 bit (36) unal ;
```

WHERE:

1. version
   is the version number of this structure

2. absolute_path
   is the absolute pathname of the in_path, with the model or submodel suffix, if the path refers to a MRDS model or submodel. If the structure is allocated, this entry will be filled in.

3. not_mrds
   is "1"b, if the path does not refer to a MRDS database model or submodel.

4. model
   is "1"b, if the path refers to a MRDS database and not a submodel

5. submodel
   is "1"b, if the path refers to a MRDS submodel, and not a database model

6. mbz1
   is reserved for future use

7. mrds_version

is the version number of the MRDS model or submodel that was found. The latest version database model is 4, and for submodels it is 5.

8. creator_id
   is the person.project.tag information returned from get_group_id_ for the person that created the database model or submodel

9. creation_time
   is the time the database model or submodel was created, in a form acceptable to date_time_

10. mbz2
   is reserved for future use

Currently, the only structure version available is 1.

5.4 GET_POPULATION

ENTRY: dsl_$get_population

This entry returns the current number of tuples in either a permanent or temporary relation.

USAGE

```
     declare dsl_$get_population entry () options (variable) ;

     call dsl_$get_population (db_index, relation_identifier,
          tuple_count, error_code) ;
```

WHERE:

1. db_index (Input) (fixed bin(35))
   is the integer returned from a call to dsl_$open, which refers to the opening for which population statistics are desired.

2. relation_identifier (Input)
   if the identification for the relation whose tuple count is to be returned. If it is declared as character, and starts with a letter, then it is interpreted as a permanent relation name. If the string does not start with a letter, and it can be converted to a number, then it will be interpreted as a temporary relation index. If the relation identifier is declared as fixed bin (35), then it is interpreted as a temporary relation index.

3. tuple_count (Output) (fixed bin(35))
   is the current tuple count for the specified relation in this openings view.

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

   mrds_error_$unknown_relation_name if the permanent relation name given is not known in this opening view of the database.

   mrds_error_$undef_temp_rel if the temporary relation index given, does not refer to a temporary relation currently defined in this opening.

   mrds_error_$invalid_db_index if the given db_index does not refer to a model or submodel opening of a database in the users process.

NOTES

This entry can be used to determine the number of tuples selected by a selection expression by defining a temporary relation using that selection expression, and calling dsl_$get_population for that temporary relation.

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

```
     declare 1 mrds_relation_list aligned
              based (mrds_relation_list_ptr),
             2 version fixed bin,
             2 access_info_version fixed bin,
             2 num_rels_in_view fixed bin,
             2 submodel_view bit (1) unal,
             2 mbz1 bit (35) unal,
             2 relation (0
              refer (mrds_relation_list.num_rels_in_view)),
               3 model_name char (32),
               3 submodel_name char (64),
               3 system_acl char (8) varying,
               3 mrds_access char (8) varying,
               3 effective_access char (8) varying,
               3 virtual_relation bit (1) unal,
               3 mbz2 bit (35) unal ;
```

WHERE:

1. version
   is the version number for this structure.

2. access_info_version
   is the version number of the access information being returned. Version 3 access_info_version is for version 3 databases with r-s-m-d MRDS relation access modes. Version 4 is for version 4 databases without attribute level security, using Multics acl's from r-e-w. Version 5 is for version 4 databases with attribute level security, using the MRDS relation access modes of append_tuple (a), and delete_tuple (d).

3. num_rels_in_view
   is the number of relations present in the view provided by this opening of the database.

4. submodel_view
   is "1", if this opening of the database was made through a submodel.

5. mbz1
   is reserved for future use

6. model_name
   is the name of this relation in the database model. If the database is secured, and the user is not a DBA, then this field will be blanks.

7. submodel_name
   is the name of this relation in the submodel view, if this opening was via a submodel. Otherwise this will be the same as the model name.

8. system_acl
   is the Multics acl's on the relation data, from the modes r-e-w.

9. mrds_access
   is the MRDS access mode for this relation. See access_info_version for the values that can be returned.

10. effective_access
    is the result of applying both Multics and MRDS access modes for this relation. This effect is returned in MRDS access values.

11. virtual_relation
    is "1"b, if the relation is defined in a submodel over more than one relation. This capability is not yet available.

12. mbz2
    is reserved for future use.

Currently, the only structure version available is 1.

5.6 GET_SCOPE

ENTRY: dsl_$get_scope

This entry returns the scope currently set on a given relation for the specified opening of the database.

USAGE

```
     declare dsl_$get_scope entry(fixed bin(35), char(*),
          fixed bin, fixed bin, fixed bin, fixed bin(35)) ;

     call dsl_$get_scope(db_index, relation_name,
          permits, prevents, scope_version, error_code) ;
```

WHERE:

1. db_index (Input) (fixed bin(35))
   is the integer returned from a call to dsl_$open, which refers to the opening for which scope information is desired.

2. relation_name (Input) (char(*))
   is the name of the relation for which scope information is desired in this opening.

3. permits (Output) (fixed bin)
   is the sum of the scope modes, representing operations that are to be permitted the caller for this relation, in this opening. See the table of scope mode encodings in the Notes below.

4. prevents (Output) (fixed bin)
   is the sum of the scope modes, representing operations that are to be denied other users of this database for this relation. See the table of scope mode encodings in the Notes below.

5. scope_version (Output) (fixed bin)
   if this value is less than 5, then the scope mode encoding for the scope represents the old operations of read - store - delete - modify, otherwise the scope mode encoding represents the new operations of read_attr - append_tuple - delete_tuple - modify_attr. with attribute level security.

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

mrds_error_$unknown_relation_name if the supplied relation name is not in the opening view specified by db_index

mrds_error_$scope_not_set if no scope is currently set on the specified relation.

NOTES

The scope modes are encoded using the integer values given below:

SCOPE CODE	OPERATION
0	null
1	read_attr or read
2	append_tuple or store
4	delete_tuple or delete
8	modify_attr or modify

See appendix F for the include file mrds_new_scope_modes.incl.pl1 giving named constants for these values.

5.7 LIST_OPENINGS

ENTRY: dsl_$list_openings

This entry returns information about all openings of MRDS databases in the users process.

USAGE

```
     declare dsl_$list_openings entry
          (ptr, fixed bin, ptr, fixed bin(35) ;

     call dsl_$list_openings (area_ptr, structure_version,
          mrds_database_openings_ptr, error_code) ;
```

WHERE:

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

2. structure_version (Input) (fixed bin)
   is the desired version of the structure that is to return opening information.

3. mrds_database_opening_ptr (Output) (pointer)
   a pointer to an allocated structure containing the opening information, which is described in the Notes below.

4. 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 hold the opening 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

NOTES

Note that the structure is still allocated, and a 0 error code returned, even if the total number of open databases is 0.

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

```
declare 1 mrds_database_openings aligned
         based (mrds_database_openings_ptr),
        2 version fixed bin,
        2 number_open fixed bin,
        2 mbz1 bit (36) unal,
        2 db (0
         refer (mrds_database_openings.number_open)),
          3 index fixed bin (35),
          3 path char (168),
          3 mode char (20),
          3 model bit (1) unal,
          3 submodel bit (1) unal,
          3 mbz2 bit (34) unal ;
```

WHERE:

1. version
   is the version number of this structure

2. number_open
   is the total number of openings for this process

3. mbz1
   is reserved for future use

4. index
   is the integer returned from a call to dsl_$open for this particular opening.

5. path
   is the absolute path of the model or submodel that was used in the call to dsl_$open for this opening. The model or submodel suffix will be present.

6. mode
   is the mode that was used in the call to dsl_$open for this opening. It can be retrieval, update, exclusive_retrieval, or exclusive_update.

7. model
   is "1", if this opening was made through the database model, and not a submodel

8. submodel
   is "1"b, if this opening was through a submodel, not a model

9. mbz2
   is reserved for future use.

Currently, the only structure version available is 1.

6.0 CHANGED DSL_ ENTRIES

6.1 SELECTION EXPRESSION ARGUMENT

The following entries will have their documentation changed for the selection expression argument as follows:

```
ENTRY:    dsl_$define_temp_rel
          dsl_$delete
          dsl_$modify
          dsl_$retrieve
```

USAGE

```
     declare dsl_$define_temp_rel entry options (variable) ;
     declare dsl_$delete entry options (variable) ;
     declare dsl_$modify entry options (variable) ;
     declare dsl_$retrieve entry options (variable) ;

     call dsl_$define_temp_rel(db_index, selection_expression,
          ...);
     call dsl_$delete (db_index, selection_expression, ... ) ;
     call dsl_$modify (db_index, selection_expression, ... ) ;
     call dsl_$retrieve (db_index, selection_expression, ... ) ;
```

WHERE:

1. db_index
   is the integer returned by a call to dsl_$open referring to the database opening that is desired for this operation.

2. selection_expression
   is a character string as defined in the section on "Selection Mechanism". It may be a constant, or a variable declared as either character, or character varying. { ... other details for the specific entries ... }

With documentation for the other entry specific arguments continuing from here.

6.2 DOCUMENTATION ONLY

The entry dsl_$get_temp_dir will refer to the companion entry dsl_$get_opening_temp_dir in it's notes section. It will make clear that the temporary storage directory path returned by the former is for the next opening to be made, while the later can obtain this information for existing openings.

All dsl_ entries that do not have examples will have examples of their use added to the manual. Existing examples will be reviewed, and corrected or expanded if necessary.

Each data manipulation entry (dsl_$store, dsl_$delete, dsl_$modify, and dsl_$retreive) will have shared opening scope requirements added to their documentation. Access requirements of attribute level security will also be added.

6.3 SCOPE CHANGES

The entries dsl_$dl_scope, dsl_$set_scope, and dsl_$set_scope_all will have their scope mode encoding tables changed to look like that given in the documentation for the new entry dsl_$get_scope. Also the include file reference for obtaining named constants for the encodings will be changed to the new include file mrds_new_scope_modes.incl.pl1. All examples making use of scope codes will be changed to reflect the new meanings. Examples showing partial deletion of scope will be added.

Each scope setting entry will have the following table of Multics acl and MRDS access requirements added to their documentation.

Access requirements on the relation (s) for which scope is being set in terms of Multics acl's, and MRDS access modes are as follows:

REQUESTED PERMIT	RELATION MSF ACL	MRDS ACCESS
a	rw	a
d	rw	d
m	rw	m on some attr in the relation
r	r	r on some attr in the relation
n	r	n

6.4 OPEN

The changes documented here will be the effects that the new security approach has on opening a database, and the necessity of converting from the old to the new scope modes. These changes are also documented in [4].

The following will be added to the NOTES section for the documentation for dsl_$open:

NOTES

If the database being opened has been secured, then the view_path must refer to a submodel that resides in the databases "secure.submodels" directory under the database directory. These must be version 5 submodels if attribute level security is to be provided. See secure_mrds_db, and the appendix on security.

If the database being opened uses a version 4 concurrency control, then adjust_mrds_db with the -reset option must be run against it, to update it to version 5 concurrency control, before it can be opened. This changes the scope modes from r-u, to read_attr, modify_attr, append_tuple, delete_tuple.

Application programs calling dsl_$set_scope, dsl_$set_scope_all, or dsl_$dl_scope making use of r-s-m-d encodings will not be impacted. Those programs using the r-u encodings will have to be changed to the encodings given in this manual.

Access requirements for all opening modes includes "r" acl on the db_model segment and relation model segments (these segments have a ".m" suffix) for any relations appearing in the given view, plus "rw" acl on the database concurrency control segment. Unshared openings require that for any relation appearing in the view, the multi-segment file containing the data must have "r" acl for exclusive_retrieval or "rw" acl for exclusive_update opening mode. For attribute level security, exclusive_retrieval mode requires read_attr on some attribute in each relation in the opening view, and exclusive_update mode requires one of append_tuple on the relation, delete_tuple on the relation, or modify_attr on some attribute in the relation, for each of the relations in the opening view.

See the examples for the mrds_call function open.

These examples referenced in these additions to the NOTES section for dsl_$open can be found in the mrds_call changes listed in [4].

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 to the MRDS Command Interface,
MTB-503

[5] Changes to the MRDS dmd_ Subroutine Interface,
MTB-505

[6] Extensions to the create_mrds_dsm and display_mrds_dsm
Commands for MRDS security.
MTB-506

[7] Changes in the MRDS Submodel Interface
MTB-496

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 to the MRDS Command Interface, MTB-503

[5] Changes to the MRDS dmd_ Subroutine Interface, MTB-505

[6] Extensions to the create_mrds_dsm and display_mrds_dsm Commands for MRDS security. MTB-506

[7] Changes in the MRDS Submodel Interface MTB-496