To: MTB Distribution

From: Jim Gray

Date: March 31, 1981

Subject: Changes to the MRDS Command Interface.

Send comments by:

Multics mail on System M to JGray.Multics

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

Continuum meeting MRDS_development, link to transaction 352

Multics Project internal working documentation. Not to be reproduced outside the Multics Project.

1.0 INTRODUCTION

This MTB discusses in detail, the changes to be made to commands used in the Multics Relational Data Store (MRDS), in the next release (MR9). Changes to the subroutines interfaces to MRDS are the topics of other MTB's. (see the reference section)

The organization of the paper is by individual command, with new commands first. The format used is that of user documentation, as it will appear in the new MRDS manual, with the descriptions here replacing any existing descriptions in the current version of the manual.

The following new commands are included:

LONG NAME	SHORT NAME
display_mrds_db_access	dmda
display_mrds_db_population	dmdp
display_mrds_scope_settings	dmss
secure_mrds_db	smdb
unpopulate_mrds_db	umdb

The following existing commands are covered:

LONG NAME	SHORT NAME
adjust_mrds_db	amdb
create_mrds_db	cmdb
create_mrds_dm_include	cmdmi
create_mrds_dm_table	cmdmt
create_mrds_dsm	cmdsm
display_mrds_db_status	dmdbs
display_mrds_dm	dmdm
display_mrds_dsm	dmdsm
display_mrds_open_dbs	dmod
mrds_call	mrc
quiesce_mrds_db	qmdb
update_mrds_db_version	umdv

1.1 BACKGROUND INFORMATION

The motiviation for the majority of changes to MRDS have been for one of three reasons.

The most important one is that of providing attribute level security for the next release of MRDS. This is to be provided by forcing all users to open through a new version of the submodel that contains access control information, once the database has been secured.

A second major change to MRDS has been the removal of the MR8 release concurrency control modes of read-update, and their replacement with attribute level security compatible modes of read_attr, modify_attr, append_tuple, delete_tuple.

The third motivation for changes to the MRDS interface was the adoption of a true Database Administrator (DBA) concept. A DBA will be defined in the new MRDS manual as a person holding "sma" access on the database directory. He will be the only person allowed to create the new secure submodels. Once the database is secured, he will be the only person that can use the model view, rather than a secured submodel view, of the database. Certain security sensitive, or administrative commands will be restricted to be usable by only this DBA.

The first two changes caused some minor changes to the database structure. The first was the addition of a "secure.submodels" directory underneath the database directory, where all submodels that are to provide attribute level security are to be placed, so that the database manager can maintain control over them. The second was the renaming of the database concurrency control segment from "dbc" to "db.control". This was done to allow a proper transition by the user from the old scope (concurrency control) modes to the new one. Details on the second change can be found in the approved MCR 4812.

The remaining changes came about because of SCP items, and TR's such as 7044, 7072, 7144, 7163, 7605, 7780, 7841, 7868, 8133, 8424, 8588, 8990, and 8991.

1.2 SUMMARY OF CHANGES

The following gives a brief idea of what changes were made to each command, and for what reason.

The mrds_call changes are listed by function, using their short names in the order: delete_scope, get_population, get_scope, list_dbs, open, set_scope, set_scope_all.

SHORT NAME	CHANGES	REASON
amdb	converts to new scope modes	see MCR 4812
cmdb	-secure option, secure.submodels dir	attribute level security
cmdmi	-based option, secure submodel view	attribute level security + SCP
cmdmt	secure submodel view	attribute level security
cmdsm	attribute access modes	attribute level security
dmda	display new access	attribute level security
dmdbs	secure submodel view	attribute level security
dmdm	revise user tuple view	answer to TR 7163
dmdsm	display new access	attribute level security
dmod	extend info, version handing	answer to TR 8424
dmdp	display tuple counts	answer users suggestion TR 8990
dmss	display new scope, opening info	concurrency control changes
qmdb	restrict to DBA	attribute level security
smdb	DBA tool to secure database	attribute level security
umdb	database development tool	desired by users
umdv	won't work on secured db	attribute level security

MRDS
CALL

ds	new scope modes	concurrency control changes
gp	get tuple count	users suggestion TR 8991
gs	display scope	replace lost function, TR 7780
ld	extend openings handled	fix TR 8424
o	open only via secured submodel	attribute level security
ss	new scope modes	concurrency control changes
ssa	new scope modes	concurrency control changes

2.0 NEW MRDS COMMANDS

These commands have not previously appeared in a release of MRDS.

2.1 DISPLAY_MRDS_DB_ACCESS

This command is the primary access display tool to be used by MRDS in it's new security approach. It also replaces lost functionality of the dsmd_$validate_rel interface, as well as showing the new security access modes that are to be used.

NAME: display_mrds_db_access, dmda

This command displays the current access that the user has to the data for the relations in the supplied view of the database.

USAGE

display_mrds_db_access path {-control_args}

WHERE:

1. path
   is the relative or absolute pathname of a database model or submodel, with or without suffix, that supplies the view for which the user wishes to see access information. A suffix will be required for models and submodels having the same name, and residing in the same directory. If none is given, the model will be found before the submodel.

2. control_args
   may be one of the following:

   -brief, -bf
   specifies that a short form of the access information is to be displayed, showing only effective access to the data.

   -long, -lg
   specifies that all information related to access be displayed. This is the default.

   -relation rel_name1 ... rel_nameN
   specifies that only the access for those relations whose names are given in the rel_nameI list is to be displayed according to the other control arguments.

NOTES

If the database has been secured, then path must refer to a secure submodel. The user must have sufficient access to the related model information to open the database using the given path.

The Multics system acl's, the MRDS access modes, and the result of these two, an effective access, is displayed for each relation and attribute in the given view. MRDS access and effective access use the following version dependent modes:

DB VERSION	SM VERSION	MODES
<=3	any	r-s-m-d
4	<=4	r-e-w
4	5	r-a-m-d

where the database and submodel versions are part of the long form of the display, or can be found using the command display_mrds_db_version. Openings through the model use the access modes for the first two entries in the table, depending on the database version. The r-e-w refers to Multics acl's. The r-s-m-d refers to the operations of read, store, modify, delete. The r-a-m-d refers to the new attribute level security related operations of read_attr, append_tuple, modify_attr, and delete_tuple.

EXAMPLES

```
! display_mrds_db_access submodel

Database path: >udd>m>jg>dr>model.db
      version: 4

Submodel path: >udd>m>jg>dr>submodel.dsm
      version: 5

Relation  Attribute          System    MRDS      Effective

r001                           r         a          n
          k001                 r         r          r
          d001                 r         m          n

r002                           rw        d          d
          k001                 rw        m          m
          x001                 r         n          n
          d001                 n         r          n


! display_mrds_db_access submodel -brief

r001                           n
          k001                 r
          d001                 n

r002                           d
          k001                 m
          x001                 n
          d001                 n
```

2.2 DISPLAY_MRDS_DB_POPULATION

This command has been on the users request list for a long time. It is a current tool being extended to submodel views, to make it usable for secured databases.

NAME: display_mrds_db_population, dmdp

This command displays the current tuple count for each relation in the given database model or submodel view. It can also display population statistics about the vfile for each relation's data.

USAGE

display_mrds_db_population path {-control_arg}

WHERE:

1. path
   is the relative or absolute pathname of the database model or submodel, with or without suffix, that is to have the relation's population statistics displayed. A suffix will be required for models and submodels having the same name, and residing in the same directory. If none is given, the model will be found before the submodel.

2. control_args
   may be the following:

   -brief, -bf
   this limits the output display to only relation names and their current tuple count. This is the default.

   -long, -lg
   causes all statistics available via vfile_status_ to be displayed for each relation in the view.

   -relation rel_name1 ... rel_nameN
   specifies that only the population for those relations whose names are given in the rel_nameI list are to be displayed according to the other control arguments.

NOTES

Version 3 databases must have been opened at least once for exclusive update. They can not have secondary index information displayed.

The user must be a DBA to use this command on a secured database, using the model view. The user must have at least "r" access to the relation model segment, and the relation data vfile, for those relations in the view presented by "path".

The long form of the statistics displays all data available via vfile_status_ for each relation. See the documentation on that subroutine for indexed files in the MPM Subroutines Reference Manual, for more details of the output.

EXAMPLES

```
! display_mrds_db_population test -bf

Opening version 4 data model: >udd>m>jg>dr>test.db

   RELATION                                TUPLES

     r001                                    100
     r002                                    100
```

```
! display_mrds_db_population  x001  -long

Opening version 4 data model:  >udd>m>jg>dr>x001.db

Relation:  r001
  Tuples:  100
   Bytes:  2000

          Vfile version:  40/41
             vfile keys:  200          bytes:  1400
               dup keys:  95           bytes:  665
          b-tree height:  1            pages:  1
             free space:  1          updates:  201
```

The following describes the long form of the output display:

Relation: is the name of the relation in the users view

Tuples: the number of tuples currently stored as records in the vfile.

Bytes: The total number of characters used by the tuple records in the vfile_

Vfile version: file/program version for the vfile

keys: the number of vfile keys used for all tuples, there will be one for each secondary index in each tuple, and one for each primary key in each tuple.

key bytes: the total number of characters used for all keys

dup keys: is the number of secondary index attribute tuple instances that contain duplicate values.

dup key bytes: the total number of characters used by dup keys

b-tree height: the height (number of levels) currently used for storing all keys in the b-tree used for key and index accesses to the data records (tuples).

pages: is the number of pages currently used to store the b-tree

free space: is the number of pages in the record space that are part of the vfile, but currently unused by any record (tuple)

updates: is the number of times the vfile has been changed by a delete, store, or modify operation.

2.3 DISPLAY_MRDS_SCOPE_SETTINGS

This command was written because MRDS did not have a convenient interface to accomplish this task. It replaces some of the lost functionality of dsmd_$validate_rel, which was removed from MRDS in the last release. It shows the changes in the scope mode names that will be used in the coming release of MRDS, as part of the security work changes. It also shows opening information, not otherwise available, such as both the model and submodel paths used in a submodel opening.

NAME: display_mrds_scope_settings, dmss

This command will display concurrency control scope mode information for all currently open databases in the users process. The versions of the concurrency control, database, and submodel (if used for the opening) are displayed, as well as the absolute paths of the database, and the submodel (if used for the opening). The opening mode is also displayed.

USAGE

display_mrds_scope_settings

NOTES

All versions of database scope settings may be displayed, with r-s-m-d modes being used for version 3 and earlier databases, and read_attr, modify_attr, append_tuple, and delete_tuple (abbreviated as r-m-a-d) being used for version 4 databases with version 5 concurrency control. (see the notes section of adjust_mrds_db on version 5 concurrency control, this is not the same as version 5 submodels)

EXAMPLES

```
! mrds_call set_modes no_list
! mrds_call open mod_del u
! mrds_call set_scope_all 1 ru ru
! mrds_call open view eu
! display_mrds_scope_settings

Scope settings for process:  JGray.Multics.a
           process number:  4720336407

Opening index:  1
         mode:  update

          Concurrency control version:  5
                  database model path:  >udd>m>jg>dr>mod_del.db
                     database version:  4

          Relation                              Permits   Prevents

          r001                                    ramd      ramd
          r002                                    ramd      ramd

Opening index:  2
         mode:  exclusive_update

          Concurrency control version:  5
                  database model path:  >udd>m>jg>dr>partial.db
                     database version:  4

                  Opened via submodel:  >udd>m>jg>dr>view.dsm
                     submodel version:  5

          Relation                              Permits   Prevents

          part                                    ramd      ramd
          reorder                                 ramd      ramd
```

2.4 SECURE_MRDS_DB

This command was written to satisfy the needs of the new approach to security for MRDS, as outlined in [1].

NAME: secure_mrds_db, smdb

This command provides the ability to turn on (or off) the new attribute level security control features of MRDS. This is done on a database basis. The secured state of a database can also be displayed by this command.

USAGE

secure_mrds_db db_path {-control_args}

WHERE:

1. db_path
   is the relative or absolute pathname of the database to be secured, un-secured, or have it's secured state displayed. The database suffix need not be given. The path must be to a version 4 database, not a submodel.

2. control_args
   may be chosen from one of the following:

   -set
   causes the specified database to be secured, regardless of it's current secured state. This is the default.

   -reset, -rs
   causes the specified database to be un-secured, regardless of it's current secured state.

   -display
   causes the current database secured state to be displayed without affecting that state.

NOTES

A database that has been secured can only be opened via a submodel residing in the "secure.submodels" directory underneath the database directory. The purpose of this is to allow turning on(or off) attribute level security, which is implemented via submodel views, using their new access control modes (version 5 submodels). Databases earlier than version 4 are not supported.

This command requires the user to be a DBA. Commands that normally operate against the model view will require the user to be a DBA, once the database has been secured. Commands using a

submodel view will require non-DBA's to use secured submodels, once the database has been secured.

See the documentation for create_mrds_db -secure, create_mrds_dsm -install, mmi_$get_secured_state, mmi_$get_authorization, and the appendix on security.

EXAMPLES

```
! secure_mrds_db foo
The database at ">udd>m>jg>dr>foo.db" has been secured.
! secure_mrds_db foo -display
The database at ">udd>m>jg>dr>foo.db" has been secured.
! secure_mrds_db foo -reset
The database at ">udd>m>jg>dr>foo.db" is not secured.
```

2.5 UNPOPULATE_MRDS_DB

This is primarily a database application development tool. It had been written for test purposes, but was also requested by MRDS users. It is restricted to use by a DBA.

NAME: unpopulate_mrds_db, umdb

This command deletes all existing data stored in the given data base, returning it to the unpopulated state. It is primarily a data base application development tool.

USAGE

unpopulate_mrds_db database_path {-control_args}

WHERE:

1. database_path
   is the relative or absolute pathname, with or without suffix, of the database, that is to have all tuples in all relations deleted.

2. control_args may be one of the following:

   -no_force, -nfc
   causes the user to be queried as to whether he really wishes to delete all data in the database, as a safety measure against inadvertently typing in the wrong database name. This is the default.

   -force, -fc
   causes the data to be deleted without querying the user

NOTES

The user must be a DBA to use this command.

If there is no data in the data base, no error will be issued.

The command display_mrds_db_population can be used to check the current tuple count of the relations.

EXAMPLES

```
.! display_mrds_db_population test -bf

Opening version 4 data model: >udd>m>jg>dr>test.db

   RELATION                                TUPLES

     r001                                    100
     r002                                    100

! unpopulate_mrds_db test

unpopulate_mrds_db: Do you really wish to delete all data
currently stored in the database ">udd>m>jg>dr>test.db"?
! yes

Opening version 4 database: >udd>m>jg>dr>test.db

Data deletion complete, closing database.

! display_mrds_db_population test -bf

Opening version 4 data model: >udd>m>jg>dr>test.db

   RELATION                                TUPLES

     r001                                      0
     r002                                      0
```

3.0 COMMANDS THAT HAVE CHANGED

These commands have been revised to fix bugs, improve features, and provide for the new security work.

3.1 ADJUST_MRDS_DB

The primary change to this command involves the change to the concurrency control version of the database, and the use of new scope modes for this release. This command has the job of reformatting the database control segment from the old r-u (read, update) to the new r-a-m-d (read_attr, append_tuple, modify_attr, delete_tuple) scope modes.

NAME: adjust_mrds_db, amdb

This is a DBA tool for handling special problems that may arise involving the database concurrency control segment. It may be used to re-establish consistency in concurrency control after an incomplete database operation has put the database in a potentially invalid state. It may also be used to remove dead process information from the control segment, or change the setting of the concurrency control trouble switch.

USAGE

adjust_mrds_db db_path {-control_args}

where:

1. db_path
   is the relative or absolute pathname of the database, whose concurrency control segment is to be manipulated. The ".db" suffix need not be given for new version databases. This can not be a submodel pathname.

2. control_args
   may be chosen from the following:

   -reset, -rs
   the database control segment is re-established in a consistent state. If there are active users of the database, the command will query the invoker as to whether to continue, since these active users will lose their concurrency control protection if he proceeds. This control_arg is the default.

   -force, -fc
   causes the query given for the -reset control argument to be suppressed.

-no_force, -nfc
allows the query for the -reset control argument to be given. This is the default.

-dead_procs, -dpr
the database control segment has any information pertaining to dead processes (database openers, whose processes terminated without closing the database) removed. Non-passive dead processes (processes with some form of update scope set), may have left the database in an inconsistent state.

-trouble_switch state, -tsw state
where state may be either "on" or "off". This causes the database concurrency control trouble switch to be set on or off. If this switch is on, attempts to open the database will fail. This can be used to lock out database users, when there is a question about the database integrity.

NOTES

The -reset and -dead_proc options may not be used together. The -force and -no_force control arguments have no effect, except when used with the -reset option. The user must be a DBA to use this command.

The -reset option (the default) should be used only after display_mrds_db_status has been invoked to determine if there are open users, and those users are notified to close their opening of the database. If open users are active during use of this option, they will lose their concurrency control protection, and later inconsistencies may arise.

The use of the -reset option causes version 4 concurrency control, using the read-update scope modes, to be updated to version 5 concurrency control using the scope modes read_attr, modify_attr, append_tuple, and delete_tuple. Version 5 concurrency control uses a segment named "db.control", rather than "dbc". Version 4 concurrency control can not be used with the current version of MRDS, and adjust_mrds_db with the -reset option must be used on the database in order to convert it to version 5 concurrency control. The current version of concurrency control may displayed via display_mrds_db_status, using the -long option.

Current users of r-s-m-d scope mode encodings will not have to change their application programs to use version 5 concurrency control. Application programs calling dsl_$set_scope or dsl_$set_scope_all, that use the old r-u scope mode encodings, will need to change to the encodings given in this manual.

EXAMPLES

```
! display_mrds_db_status dmdm -long

Concurrency control version: 4
            Data base path: >udd>Multics>JGray>dr>dmdm.db
                   Version: 4
                     State: Consistent
                Open users: 0

! mrds_call open dmdm update

Error: mu_concurrency_control error by >unb>bound_mrds_|2232 The
database is a version not supported by this command/subroutine.
The version of the control segment has changed, to support
r-m-a(s)-d instead of r-u scope modes. "adjust_mrds_db
>udd>m>jg>dr>dmdm.db -reset" must be run before it can be used.

mrds_call: The database is a version not supported by this
command/subroutine. (From dsl_$open)

! adjust_mrds_db dmdm -reset
! mrds_call open dmdm update

Open data base is:
1         >user_dir_dir>Multics>JGray>dr>dmdm.db
          update


! display_mrds_db_status dmdm -long

Concurrency control version: 5
            Data base path: >udd>m>jg>dr>dmdm.db
                   Version: 4
                     State: Consistent
                Open users: 1

               Scope users: 0 Active
                            0 Awakening
                            0 Queued

             User process id: JGray.Multics.a
              Process number: 007720037664
               Process state: Alive
                  Usage mode: Normal
                       Scope: None

! adjust_mrds_db dmdm

adjust_mrds_db: There are open users who may be harmed if you
reset. Do you still wish to reset the >udd>m>jg>dr>dmdm.db data
base??
```

```
! no

! display_mrds_db_status  cmdmi_2

            Data base path:  >udd>m>jg>dr>cmdmi_2.db
                Open users:  1

               Scope users:  1  Active

            User process id:  JGray.Multics.a
              Process state:  Dead

                  Relation                                Permits    Prevents

                    rel_1                                  ramd       ramd
                    rel_2                                  ramd       ramd

! adjust_mrds_db  cmdmi_2  -dead_procs

! display_mrds_db_status  cmdmi_2

            Data base path:  >udd>m>jg>dr>cmdmi_2.db
                Open users:  0
```

3.2 CREATE_MRDS_DB

There have been some minor bug fixes, such as disallowing relation names that are used as segments in the new MRDS architecture. The command now gives the version of the database being created when it is invoked. The major changes are the addition of the -secure option, to have the database created in a secure state, and the removal of the reserved keywords restriction. The database architecture has also changed, in that the concurrency control segment is now named "db.control", and there is an addition of a "secure.submodels" directory.

NAME: create_mrds_db, cmdb

This command creates an unpopulated MRDS data base from a data model source segment.

USAGE

create_mrds_db source_path {database_path} {-control_args}

WHERE:

1. source_path
   is the pathname of a data model source segment. If source_path does not have a suffix of cmdb, then one is assumed. However, the cmdb suffix must be the last component of the name of the source segment.

2. database_path
   is the pathname of the data base to be created. If database_path is not given as an argument then the data base is created in the working directory with the same name as the source segment with a db (rather than a cmdb) suffix. If database_path is given as an argument then db suffix is added automatically if not given with the argument. See the notes on the structure of the database.

3. control_args
   may be chosen from the following:

   -list, -ls
   a segment containing a listing of the data model source, followed by detailed information about each relation and attribute in the resulting data base. This segment is created in the working directory and has the same name as the source segment with a list (rather than cmdb) suffix.

-no_list, -nls
indicates that no listing is to be created. This is the default.

-temp_dir path
provides for a directory with more quota than the default of the process directory when more temporary storage is needed to do a create_mrds_db on a source with many relations and attributes. For example, doing a create_mrds_db on a 127 relation source requires this argument. If the user gets a record quota overflow in the process directory during a create_mrds_db, then a new_process is required. A retry of the create_mrds_db with the -temp_dir argument, giving a pathname of a directory with more quota than the process directory, can then be done.

-secure
causes the database to be created in the secured state. See the secure_mrds_db command for details on the secured state. Also refer to the security appendix for information on the effect of the secured state on commands and subroutines.

-no_secure
causes the database to be created in the un-secure state. This is the default.

-force, -fc
causing an existing database of the same pathname as the given, or default pathname, to be deleted, and this new database created in it's place.

-no_force, -nfc
does not allow a database of the same pathname as the given or default pathname, to be created, when such a database already exists. This is the default.

NOTES

Error messages are written to the error_output I/O switch as they occur. They are also included in the listing segment if one is produced.

The database may be populated via dsl_$store, mrds_call store, or LINUS store, after the database has been opened, by the corresponding open routine. (to use LINUS, refer to the Logical Inquiry and Update System Reference Manual)

The person that invokes the create_mrds_db command automatically becomes a DBA for the database created. This is

because the creator of a database is always given "sma" access to the database directory.

DATA MODEL SOURCE SYNTAX

The basic format for a text segment containing source for the create_mrds_db command is as follows:

```
domain :
          domain_name_1  declaration_1  {options_1},
                    .
                    .
          domain_name_N  declaration_N  {options_N} ;

attribute :
          attribute_name_1  attribute_1_domain_name,
                    .
                    .
          attribute_name_N  attribute_N_domain_name ;

relation :
          relation_name_1 (
               rel_1_key_attr_1*  ... rel_1_key_attr_J*
               rel_1_data_attr_1 ... rel_1_data_attr_K  ),
                    .
                    .
          relation_name_N (
               rel_N_key_attr_1* ... rel_N_key_attr_I*
               rel_N_data_attr_1 ... rel_N_data_attr_P ) ;

index :
          indexed_relation_1_name (
               i_rel_1_i_attr_1 ... i_rel_1_i_attr_L),
                    .
                    .
          indexed_relation_N_name (
               i_rel_N_i_attr_1 ... i_rel_N_i_attr_M) ;
```

Note that the domain, attribute, relation and index statements are terminated by semicolons, while individual domain, attribute, or relation name definitions are separated by commas.

STATEMENT USAGE

The attribute and index statements are optional. The domain statement causes an attribute of the same name as the domain to be created, which can then be referenced in the relation and index statements. Additional attributes, of different names, using the existing domains can be defined via the attribute statement. The ordering of the domain, attribute, relation, and index statements must be as given, and each statement can appear at most once.

The relation statement takes previously defined attributes, and defines the relations that are to exist in the database. There must be at least one key attribute, whose purpose is to hold data values uniquely identifying each tuple to be stored in the relation. Key attributes are denoted by an asterisk after their name, in this statement only. The maximum number of key attributes is determined by the sum of the storage lengths of the individual attributes that are defined as the key attributes, known collectively as the primary key. This primary key must be less than 2277 bits. (see Section 2, User's Guide, "Data Base Design") There may be up to a total of 1000 different key and non-key attributes in any one relation. There may be up to 127 different relations defined.

The index statment is used to define attributes in previously defined relations, as being "inverted", or usable as secondary indexes. An attribute that is so defined will allow faster retrieval performance using that attribute in selection criteria, but this use increases update costs, and storage overhead for that attribute. (see Section 2, User's Guide, "Data Base Design") The same key length restrictions apply to each single inverted attribute, as apply to the total primary key.

The domain statement defines the data type that any attribute defined over that domain is to have. Any legal pll scalar data type that can be declared using the following declaration description words is allowed in MRDS.

```
aligned
binary or bin
bit
character or char
complex or cplx
decimal or dec
fixed
float or floating
nonvarying
precision or prec
real
varying or var
unaligned or unal
```

The maximum string length is 4096. Varying strings are stored at current length rather than maximum length. Refer to Appendix D of the MPM Reference guide for a description of Multics data types. When data needs to be converted from the users type into the storage type declared in the domain statement, the subroutine assign_ is used. See MPM Subsystem Writers Guide for a description of data types supported by that routine.

Relation, attribute, and domain names must start with an alphabetic character, and can be composed of any alphanumeric

character plus underscore and hyphen characters. The maximum name length is 30 characters for relation names, and 32 characters elsewhere. The names "dbc" and "db_model" are reserved, and may not be used for relation names.

FORMATTING DATA MODEL SOURCE

The keywords domain, attribute, and relation may abbreviated as dom, attr, and rel respectively.

Comments appear in the source text in the same manner that they appear in a PL/I program.

The source may be formatted in several ways, such as giving the source segment an add_name with .pl1 suffix and using indent, or creating the data base first, then capturing the output of display_mrds_dm using the -cmdb option.

DOMAIN OPTIONS

The domain statement options_I may be one or more of the following:

-check_procedure path, -check_proc path
specifies a procedure that performs data verification checks upon storage into the data base (such as ensuring valid dates). Path must be an absolute pathname.

-encode_procedure path, -encode_proc path
specifies a procedure that performs data encoding (such as the names of the states of the USA to integers 1-50) before storage into an internal data base form. Path must be an absolute pathname.

-decode_procedure path, -decode_proc path
specifies a procedure that performs data decoding upon retrieval from the data base, normally the inverse of the encode procedure. Path must be an absolute pathname.

-decode_declare DECLARE, -decode_dcl DECLARE
specifies that DECLARE is of the same form as in declaration_I in the domain statement that gives the data type to be used for the users view, and the decode procedure if present. If this option is not given then the decode procedure data type is that given in the main declaration.

See "Administrator Written Procedures" (in a later section), for a detailed explanation of the interface, and examples of how these options may be used.

DATA BASE ARCHITECTURE

The database created is a directory with the identifying suffix ".db". This directory contains the following:

NAME	TYPE	PURPOSE
db.control	segment	concurrency control
db_model	segment	domain info, relation names
{relation_name}.m	segment	model of the relation structure
{relation_name}	multi-segment file	relation data storage
secure.submodels	directory	place for secure submodels

There is one relation model segment, and one relation data msf for each relation defined in the database.

EXAMPLES

```
! print x.cmdb

      >udd>m>jg>dr>x.cmdb02/27/81  1157.2 mst Fri

dom: a bit;  /* simplest possible database */
rel: b(a*);

! create_mrds_db  x  >udd>d>dbmt>small  -list

CMDB Version 4 models.

! print  x.list

          CREATE_MRDS_DB LISTING FOR >udd>m>jg>dr>x.cmdb
          Created by:          JGray.Multics.a
          Created on:          02/27/81  1203.0 mst Fri
          Data base path:      >udd>d>dbmt>small.db
                 Options:      list


     1    dom: a bit;
     2    rel: b(a*);

NO ERRORS

DATA MODEL FOR DATA BASE  >udd>d>dbmt>small.db

Version:                4
Created by:          JGray.Multics.a
Created on:          02/27/81  1203.0 mst Fri

Total Domains:          1
Total Attributes:       1
Total Relations:        1


RELATION NAME:   b

Number attributes:           1
Key length (bits):           1
Data Length (bits):          1

   ATTRIBUTE:

      Name:        a
      Type:        Key
      Offset:      0 (bits)
      Length:      1 (bits)
      Domain_info:
```

```
          name: a
          dcl:  bit (1) nonvarying unaligned

!  print  states.cmdb

      >udd>m>jg>dr>states.cmdb       02/27/81  1207.3 mst Fri


domain :
         text char(4096) varying,
         date_time fixed bin(71)
                   -check_proc >udd>m>jg>dr>verify_date,
         dollars fixed decimal(59, 2) unal,
         state_name fixed bin -decode_dcl char(30)
                   -decode_proc >udd>m>jg>dr>convert_num_to_char
                   -encode_proc >udd>m>jg>dr>convert_char_to_num,
         vector complex float bin(63), /* longitute+latitute */
         key bit(70), /* use unique_bits_ for key values */
         name char(32) ;

attribute:
         first_name name,
         last_name name,
         salary dollars,
         expenses dollars ;

relation:
         person (last_name* first_name* salary expenses),
         state_history(key* state_name date_time text),
         person_state (last_name* first_name* key*),
         state_location(key* vector) ;

index:
         state_history(state_name) ;


!  create_mrds_db  states

CMDB Version 4 models.

!  display_mrds_dm  states


RELATION:      person
   ATTRIBUTES:
      last_name                          Key
          char (32)
      first_name                         Key
          char (32)
      salary                             Data
          fixed dec (59,2) unal
```

```
      expenses                              Data
          fixed dec (59,2) unal


RELATION:      person_state
   ATTRIBUTES:
      last_name                             Key
          char (32)
      first_name                            Key
          char (32)
      key                                   Key
          bit (70)


RELATION:      state_history
   ATTRIBUTES:
      key                                   Key
          bit (70)
      state_name                            Data  Index
          char (30)
      date_time                             Data
          fixed bin (71)
      text                                  Data
          char (4096) var


RELATION:      state_location
   ATTRIBUTES:
      key                                   Key
          bit (70)
      vector                                Data
          cplx float bin (63)
```

3.3 CREATE_MRDS_DM_INCLUDE/TABLE

These commands are implemented in the same module, and have similar changes, so they are included here together. The changes here are the addition of a -based option for CMDMI, per a SCP item, and the changes to make sure these commands are only used via secured submodels, once the database has been secured, if the user is not a DBA. A bug has been fixed so that the users view of the data is displayed if a -decode_dcl option was present in the CMDB source.

3.3.1 CREATE_MRDS_DM_INCLUDE

NAME: create_mrds_dm_include, cmdmi

This command is a MRDS data model/submodel display tool which creates an include segment suitable for use in accessing the data base from PL/I programs. Comments are put in the include file to indicate indexed and key attributes.

USAGE

create_mrds_dm_include path {-control_args}

WHERE:

1. path
   is the relative or absolute pathname of the data model/submodel of the data base, with or without suffix. It requires "retrieval" permission to the data model. If the database has been secured, then the path must refer to a submodel in the secure.submodels directory under the database, unless the user is a DBA. A suffix will be required for models and submodels having the same name, and residing in the same directory. If none is given, the model will be found before the submodel.

2. control_args
   may be one or more of the following:

   -page_length N, -pl N
   specifies the number of lines allowed between form-feed characters in the output segment, where N=0 or 30<=N<=127. A page length of 0 puts a form feed before each structure. Default is 59 lines.

   -order rel_namel rel_name2...rel_namei
   specifies that the structures generated for the relations whose names follow this argument are to be placed first in the output segment in the order of their names on the command line. The structures for relations not named in the ordered list are placed at the end of

the output segment in the order in which their names are defined in the data model. The names following the -order control argument are separated by spaces.

-based
specifies that the resulting include file structure declaration have the pll attribute "based".

-no_based
specifies that the resulting include file structure declaration will not have the based attribute. This is the default.

NOTES

The output is written to a segment whose name is constructed as follows:

<name of the data model/submodel>.incl.pll

with the .db or .dsm suffix replaced by .incl.pll. If the segment does not exist, it is created.

If both a data model and submodel of the same name are in a given directory, then the model is found first if no suffix is given.

If the database has been secured, and the user is not a DBA, then the "key" comment on attributes will be changed to "indexed".

If a -decode_declare option exists on an attribute's domain, then this declaration will appear in the include file, since this is the users view, and the database storage data type would not be of use.

EXAMPLES

```
! display_mrds_dm cmdmi -cmdb


 /* Created from   >udd>m>jg>dr>cmdmi.db
                    02/24/81  1406.9 mst Tue      */

domain:
          data
               real float decimal (10) aligned /* 9-bit */
               -decode_dcl
               character (20) varying aligned,
          indexed
               bit (36) nonvarying unaligned,
          key
               real fixed binary (17,0) aligned;


relation:
          sample                (key* data indexed);

index:
          sample                (indexed);



!  create_mrds_dm_include cmdmi -based
!  pr cmdmi.incl.pl1

/* ************************************************
   *                                              *
   * BEGIN cmdmi.incl.pl1                         *
   *       created: 02/24/81  1407.2 mst Tue      *
   *            by: create_mrds_dm_include (3.0)  *
   *                                              *
   * Data model >udd>m>jg>dr>cmdmi.db             *
   *       created: 02/24/81  1405.1 mst Tue      *
   *       version: 4                             *
   *            by: JGray.Multics.a               *
   *                                              *
   ************************************************ */

dcl 1 sample aligned based,
      2 key real fixed binary (17,0) aligned,     /* Key */
      2 data character (20) varying aligned,
      2 indexed bit (36) nonvarying unaligned;    /* Index */

/* END of cmdmi.incl.pl1    *********************************/
```

```
! display_mrds_dm cmdmi_2 -cmdb

 /* Created from   >udd>m>jg>dr>cmdmi_2.db
                    03/23/81  1417.3 mst Mon      */

domain:
          char
               character (1) nonvarying unaligned,
          number
               real float decimal (10) unaligned ;


relation:
          rel_1                (char*),
          rel_2                (number*);


! create_mrds_dm_include cmdmi_2 -order rel_2 rel_1
! print cmdmi_2.incl.pl1

/* ************************************************
   *                                              *
   * BEGIN cmdmi_2.incl.pl1                       *
   *       created: 03/16/81  1321.1 mst Mon      *
   *            by: create_mrds_dm_include (3.0)  *
   *                                              *
   * Data model >udd>m>jg>dr>cmdmi_2.db           *
   *       created: 03/16/81  1320.4 mst Mon      *
   *       version: 4                             *
   *            by: JGray.Multics.a               *
   *                                              *
   ************************************************ */

dcl 1 rel_2 aligned,
      2 number real float decimal (10) unaligned;      /* Key */

dcl 1 rel_1 aligned,
      2 char character (1) nonvarying unaligned;       /* Key */

/* END of cmdmi_2.incl.pl1    ***********************************/
```

3.3.2 CREATE_MRDS_DM_TABLE

NAME: create_mrds_dm_table, cmdmt

This command is a display tool which creates a pictorial representation of a MRDS data model/submodel. Each box names an attribute in the relation, giving its PL/I data type with flags indicating if it is a key attribute and/or index attribute in the relation.

USAGE

create_mrds_dm_table path {-control_args}

WHERE:

1. path
   is the relative or absolute pathname of the data model/submodel of the data base, with or without suffix. The user must have "retrieve" access to some relation model in the data base. The pathname must be the first argument. If the database has been secured, the path must refer to a submodel in the secure.submodels directory under the database, unless the user is a DBA. A suffix will be required for models and submodels having the same name, and residing in the same directory. If none is given, the model will be found before the submodel.

2. control_args
   may be one or more of the following:

   -brief, -bf
   suppresses the PL/I data type information normally displayed below the attribute name inside each box.

   -long, -lg
   causes the PL/I data type information to be displayed below each attribute name, inside each box. This is the default.

   -line_length N, -ll N
   specifies the maximum line length(in characters) available for the display of boxes across the page where 64<=N<=136. Default line length is 136.

   -order rel_namel rel_name2 ... rel_namei
   specifies that the displays generated for the relations whose names follow this argument are to be placed first in the output segment in the order of their names on the command line. The displays for relations not named in the ordered list are placed at the end of the output

segment in the order in which their names are defined in the data model. The names following the -order control argument are separated by spaces.

-page_length N, -pl N
specifies the number of lines allowed between new page characters in the output segment where 30<=N<=127. Default page length is 59 lines.

NOTES

The output is written to a segment whose name is constructed as follows:

<entryname of data model/submodel>.table

with the .db or .dsm suffix replaced by ".table". If the segment does not exist, it is created.

If both a data model and submodel of the same name are in the same directory, then the model is found first if no suffix is given.

If the database has been secured, and the user is not a DBA, then the key attributes will only be marked as indexed.

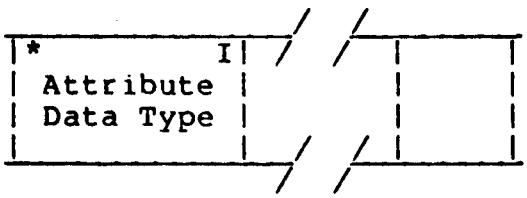
If a -decode_declare option exists on an attribute's domain, then this declaration will appear in the table, since this is the users view, and the database storage data type would not be of use.

EXAMPLES

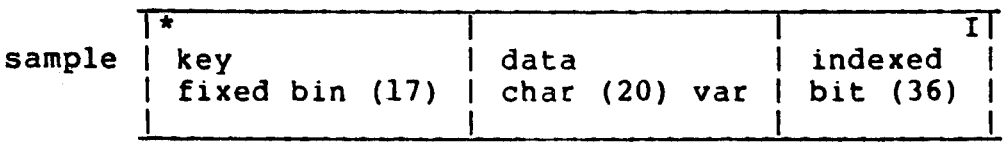
```
! display_mrds_dm  cmdmt  -cmdb

 /* Created from   >udd>m>jg>dr>cmdmt.db
                    02/26/81  1159.4 mst Thu      */

domain:
          data
               real float decimal (10) aligned /* 9-bit */
               -decode_dcl
               character (20) varying aligned,
          indexed
               bit (36) nonvarying unaligned,
          key
               real fixed binary (17,0) aligned;


relation:
          sample                (key* data indexed);

index:
          sample                (indexed);
```

```
!  create_mrds_dm_table  cmdmt  -line_length 65
!  print  cmdmt.table

/* ********************************************
   *                                          *
   * BEGIN cmdmt.table                        *
   *      created: 02/26/81  1158.6 mst Thu   *
   *           by: create_mrds_dm_table (3.0) *
   *                                          *
   * Data model >udd>m>jg>dr>cmdmt.db         *
   *      created: 02/26/81  1158.3 mst Thu   *
   *      version: 4                          *
   *           by: JGray.Multics.a            *
   *                                          *
   ******************************************** */
```

LEGEND:

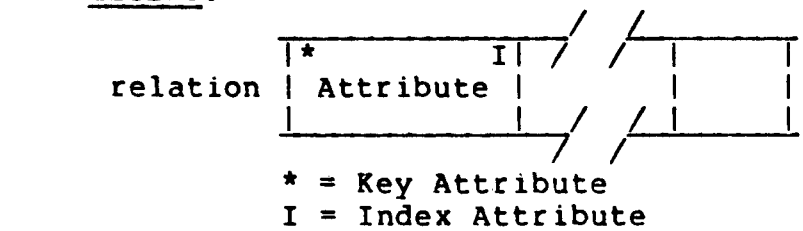


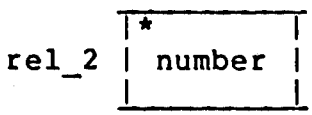
* = Key Attribute
I = Index Attribute

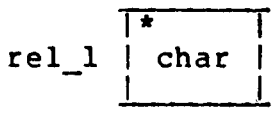


```
!  create_mrds_dm_table  cmdmi_2  -brief  -order  rel_2 rel_1
!  print  cmdmi_2.table

/* ***********************************************
   *                                             *
   * BEGIN cmdmi_2.table                         *
   *      created: 03/16/81  1342.0 mst Mon      *
   *           by: create_mrds_dm_table (3.0)    *
   *                                             *
   * Data model >udd>m>jg>dr>cmdmi_2.db          *
   *      created: 03/16/81  1320.4 mst Mon      *
   *      version: 4                             *
   *           by: JGray.Multics.a               *
   *                                             *
   *********************************************** */
```

LEGEND:







3.4 CREATE_MRDS_DSM

This command has been entirely re-written. It uses a new LRK parser, plus a syntax addition to it's source language to provide for the new security access modes. It incorporates the DBA concept, and the idea of secure submodels that are part of the security approach. The -install option is new for this purpose, causing the creation of the new secure.submodels directory in the database, if not present, and placing the newly created submodel there. The submodel specified relation and attribute names are validated against the actual database names to prevent generation of invalid submodels.

The details of this command will be documented in [5].

3.5 DISPLAY_MRDS_DB_STATUS

The work on this command is related to the security work. The command has been extended to work through submodel views. It has been restricted like CMDMI, for secured databases. The output formatting has been improved. The control argument meanings have been changed in doing this, so that -brief is really brief, the default output is different now from -brief, a kind of -terse, and no output is shown for normal conditions, only abnormal conditions, unless -long is specified. This output now shows the new scope mode names used for the security work.

NAME: display_mrds_db_status, dmdbs

This command displays the current state of the database concurrency control segment. The number and type of open users of the database can be determined from it's output. The current scope settings on all relations in the users view can be displayed.

USAGE

display_mrds_db_status path {-control_args}

WHERE:

1. path
   is the relative or absolute pathname of the database, or a submodel defined for that database, for which concurrency control information is desired. A suffix will be required for models and submodels having the same name, and residing in the same directory. If none is given, the model will be found before the submodel.

2. control_args
   may be chosen from the following:

   -brief, -bf
   causes display of only the current number of open users, and the number of active scope users of the database

   -long, -lg
   causes all possible concurrency control information to be displayed that is in the users view. This includes the concurrency control version, whether the database has been quiesced, consistency state of the database control segment, existance of any dead processes, identification of the processes having the database open and what scope they have set on relations that are in the users view.

   -user person.project, -user person

causes all possible concurrency control information (like -long) for the person.project, or person given to be displayed, including scope setting on relations in the users view.

-proc_id process_number, -pid process_number
same as for -user, but using the process number for the identifier instead.

NOTES

If no control arguments are specified, then an abbreviated form of the information given by the -long option is presented.

The output display does not include "normal" conditions, such as "Activitation: normal", only exception conditions or necessary information are displayed (e.g. "Non-passive scope set by a dead process.", or "Open users: 0") unless the -long option is specified.

EXAMPLES

```
! display_mrds_db_status 2rels -long

Concurrency control version:  5
            Data base path:  >udd>Multics>JGray>dr>2rels.db
                   Version:  4
                     State:  Consistent
                Open users:  1

               Scope users:  1  Active
                             0  Awakening
                             0  Queued

           User process id:  JGray.Multics.a
            Process number:  016600352461
             Process state:  Alive
                Usage mode:  Normal
                     Scope:  Active
                Activation:  Normal

              Relation                        Permits    Prevents

                r001                           ramd       n
                r002                           r          a


! display_mrds_db_status 2rels

            Data base path:  >udd>Multics>JGray>dr>2rels.db
                Open users:  1

               Scope users:  1  Active

           User process id:  JGray.Multics.a

              Relation                        Permits    Prevents

                r001                           ramd       n
                r002                           r          a


! display_mrds_db_status 2rels -bf

            Data base path:  >udd>Multics>JGray>dr>2rels.db
                Open users:  1

               Scope users:  1  Active
```

```
! display_mrds_db_status 2rels -person JGray.Multics

          User process id:  JGray.Multics.a
           Process number:  016600352461
            Process state:  Alive
               Usage mode:  Normal
                    Scope:  Active
               Activation:  Normal

            Relation                        Permits    Prevents

              r001                           ramd       n
              r002                           r          a
```

The following example shows the effect of using a submodel path, where that submodel references an open database "2rels.db" (see above examples), with only one relation in the submodel view. The submodel has the name "alias_1" for the model relation "r001".

```
! display_mrds_db_status  1rel.dsm

          Data base path:  >udd>m>jg>dr>2rels.db
              Open users:  1

             Scope users:  1  Active

          User process id:  JGray.Multics.a

            Relation                        Permits    Prevents

              alias_1                        ramd       ramd
```

3.6 DISPLAY_MRDS_DM

This command has had it's display of offsets within the tuple changed to be that expected by the user, rather than the internal form used by MRDS. For the brief forms of display, data types are given as the users view if a -decode_dcl option was present in the CMDB source. For a secured database, only a DBA may make use of this command.

NAME: display_mrds_dm, dmdm

This commands displays the details of the database model and data definition for a given database. It can be used to reconstruct the original create_mrds_db data model source from the database.

USAGE

display_mrds_dm db_path {-control_args}

WHERE:

1. db_path
   is the pathname of the data base for which the data model is to be displayed.

2. control_args
   may be chosen from the following:

   -brief, -bf
   specifies that only relation and attribute names are to be displayed. No information on the characteristics of the attributes and relations is provided. This control argument has no effect if specified together with -rel_names.

   -cmdb
   displays the information in a data model in a format which can be used to create another data model.

   -long, -lg
   specifies that all available information about relations and their attributes is to be displayed. For relations, this includes the number of attributes and the layout of the attributes in the tuple. For attributes, this includes the name on the underlying domain, the declaration, and bit offset and length information. This control argument has no effect if specified together with -rel_names.

-relation {rel_name_1 ... rel_name_N}
specifies that information pertaining to only the relations designated by the rel_name_I names is to be displayed. If rel_name_I is not present, information about all the relations in the data base is displayed.

-rel_names
specifies that only relation names are to be displayed.

-temp_dir path
provides for a directory with more quota than the default of the process directory when more temporary storage is needed to do a display_mrds_dm on a source with many relations and attributes. For example, doing a display_mrds_dm on a 127 relation model requires this argument. If the user gets a record quota overflow in the process directory during a display_mrds_dm, then a new_process is required. A retry of the display_mrds_dm with the -temp_dir argument, giving a pathname of a directory with more quota than the process directory should then be done.

-output_file path, -of path
specifies that the output is to be placed in the segment named by path, rather than being displayed on user_output.

-no_output_file, -nof
specifies that output is to be displayed on user_output. This is the default.

NOTES

If neither -long nor -brief is specified, the relation name is displayed for each relation, and the name and user view declaration of each attribute is displayed. If -relation is not specified, then information for every relation in the data base is displayed.

This command does not work for submodels (see display_mrds_dsm).

The user must be a DBA in order to use this command on a secured database.

If -long is specified, the header output indicates if the database has been secured.

The bit offset information starts at 0 for the first attribute in a relation and is incremented by succeeding attributes data storage bit length, plus padding that properly aligns the data on character and word boundaries per Multics data type storage standards. (see Appendix D of the MPM Reference Guide) Varying length attributes have their lengths displayed as if they were currently stored at maximum length.

EXAMPLES

```
! display_mrds_dm dmdm.db -long

DATA MODEL FOR DATA BASE  >udd>m>JGray>dr>dmdm.db
Data base secured.

Version:                4
Created by:          JGray.Multics.a
Created on:          02/24/81  1319.5 mst Tue

Total Domains:          3
Total Attributes:       3
Total Relations:        1


RELATION NAME:   sample

Number attributes:           3
Key length (bits):          36
Data Length (bits):        144

   ATTRIBUTES:

      Name:        key
      Type:        Key
      Offset:      0 (bits)
      Length:      36 (bits)
      Domain_info:
            name: key
            dcl:  character (1) nonvarying aligned

      Name:        data
      Type:        Data
      Offset:      36 (bits)
      Length:      9 (bits)
      Domain_info:
            name: data
            dcl:  character (1) nonvarying unaligned

      Name:        indexed
      Type:        Data  Index
      Offset:      72 (bits)
      Length:      72 (bits)
      Domain_info:
            name: indexed
            dcl:  character (1) varying aligned
```

```
! display_mrds_dm dmdm


RELATION:       sample
   ATTRIBUTES:
      key                                   Key
          char (1)  aligned
      data                                  Data
          char (1)
      index                                 Data  Index
          char (1) var



! display_mrds_dm dmdm -brief


RELATION:       sample
   ATTRIBUTES:          key
                        data
                        index



! display_mrds_dm dmdm -cmdb

 /* Created from   >udd>m>jg>dr>dmdm.db
                .     03/16/81  1514.1 mst Mon      */

domain:
          data
               character (1) nonvarying unaligned
               -check_proc >udd>m>jg>dr>validate_data$validate_data,
          index
               character (1) varying aligned,
          key
               character (1) nonvarying aligned;


relation:
          sample                (key* data index);

index:
          sample                (index);
```

```
! display_mrds_dm  dmdm  -rel_names
sample
```

3.7 DISPLAY_MRDS_DSM

This command has had it's output formatting revamped. It now displays the new version submodel access modes, implemented for the security work.

The details of this command will be documented in [5].

3.8 DISPLAY_MRDS_OPEN_DBS

This command has been extended to work with older version databases, and to include the opening mode in it's output.

NAME: display_mrds_open_dbs, dmod

This command displays on user_output, the database opening indexes, opening modes, and pathnames of all model and submodel openings of databases currently open in the users process.

USAGE

display_mrds_open_dbs

NOTES

The output is a formatted list of openings, in opening index order, that contains the opening model or submodel path, and the mode in which the opening was obtained. Database and submodel suffixes are shown whether they were used in the call to open or not. A ".dsm" suffix indicates the opening was through a submodel.

EXAMPLES

```
! mrds_call close -all
! display_mrds_open_dbs

No data bases are currently open.

! mrds_call open model u submodel r

Open data bases are:
1         >udd>m>jg>dr>model.db
          update
2         >udd>m>jg>dr>submodel.dsm
          retrieval

! display_mrds_open_dbs

Open data bases are:
1         >udd>m>jg>dr>model.db
          update
2         >udd>m>jg>dr>submodel.dsm
          retrieval
```

3.9 MRDS_CALL

The following functions of mrds_call have been changed, or are entirely new functions.

delete_scope
get_population (NEW)
get_scope (NEW)
list_dbs
open
set_scope
set_scope_all

3.9.1 NEW FUNCTIONS

These functions have not previously appeared in a release of MRDS.

3.9.1.1 GET_POPULATION

This function was added in answer to users suggestions. It provides a means of determining either the number of tuples in a relation or that specified by a MRDS selection expression, by obtaining the cardinality of a temporary relation defined using for obtaining permanent or temporary relation size.

FUNCTION: get_population, gp

This function returns the number of tuples that make up either a temporary or permanent relation, given the temporary relation index, or the permanent relation name. It provides a means of determining the number of tuples specified by a selection expression, by using that selection expression to define a temporary relation, and then getting it's population.

USAGE

mrds_call get_population data_base_index relation_identifier

WHERE:

1. data_base_index
   is the data base opening index displayed by the open function

2. relation_identifier
   is the identification of the relation for which the population is to be obtained. For temporary relations, it is the temporary relation index returned from a call to the define_temp_rel function. For permanent relations, it is the view relation name.

NOTE

EXAMPLES

```
! mrds_call open pop exclusive_update

Open data base is:
1         >udd>m>jg>dr>pop.db
          exclusive_update

! display_mrds_dm pop

RELATION:      r001
   ATTRIBUTES:
      k001                                Key
          fixed bin (17)
      d001                                Data
          fixed bin (17)
      x001                                Data  Index
          fixed bin (17)

! mrds_call get_population 1 r001

Tuple count:  100

! mrds_call dtr 1 "-range (r r001) -select r.k001*" 0

Temporary relation index is:  1.

! mrds_call get_population 1 1

Tuple count:  100
```

3.9.1.2 GET_SCOPE

This function was written to accomadate the new dsl_$get_scope routine, which was implemented as part of the changes to the scope modes from the old r-u, to read_attr, modify_attr, append_tuple, and delete_tuple. These changes conform with the new security access modes. In addition, this satisfies the missing scope information previously provided by the deleted dsmd_$validate_rel interface.

FUNCTION: get_scope, gs

This function provides a means of finding the current scope settings on a particular relation.

USAGE

mrds_call get_scope data_base_index relation_name

WHERE:

1. data_base_index
   is the data base opening index displayed by the open function

2. relation_name
   is the name of the relation whose scope settings are to be displayed

NOTES

The scope display uses the following abbreviations:

a	append_tuple
d	delete_tuple
m	modify_attr
n	null
r	read_attr
s	store

If the concurrency control version is less than 5, then "s" will be displayed, otherwise "a" will be used. This version can be displayed by display_mrds_db_status using the -long option, or by display_mrds_scope_settings.

EXAMPLES

```
! mrds_call open dmdm exclusive_update

Open data base is:
1         >udd>Multics>JGray>dr>dmdm.db
          exclusive_update

! display_mrds_scope_settings

Scope settings for process:  JGray.Multics.a
            process number:  2740040441

Opening index:  1
          mode:  exclusive_update

          Concurrency control version:  5
                  database model path:  >udd>m>jg>dr>dmdm.db
                     database version:  4

          Relation                              Permits   Prevents

          sample                                  ramd      ramd


! mrds_call get_scope 1 sample

Permits:  ramd      Prevents:  ramd
```

EXAMPLES

```
! mrds_call open two_rels update

Open data base is:
1         >udd>m>jg>dr>two_rels.db
          update

! mrds_call set_scope rel1 ru n rel2 r amd

! display_mrds_scope_setttings

Scope settings for process:  JGray.Multics.a
            process number:  2740040441

Opening index:  1
         mode:  update

          Concurrency control version:  5
                  database model path:  >udd>m>jg>dr>two_rels.db
                     database version:  4

          Relation                              Permits    Prevents

          rel1                                    ramd       n
          rel2                                    r          amd


! mrds_call delete_scope 1 rel1 amd n rel2 n amd

! display_mrds_scope_settings

Scope settings for process:  JGray.Multics.a
            process number:  2740040441

Opening index:  1
         mode:  update

          Concurrency control version:  5
                  database model path:  >udd>m>jg>dr>two_rels.db
                     database version:  4

          Relation                              Permits    Prevents

          rel1                                    r          n
          rel2                                    r          n
```

3.9.2.2 LIST_DBS

This function has been extended to work on old version database openings, and to work with openings not made through mrds_call.

FUNCTION: list_dbs, ld

This function displays the opening index, opening mode, and path of the submodel or model used for the opening, for all openings of MRDS databases in the users process.

USAGE

mrds_call list_dbs

NOTES

If the displayed path ends with a ".dsm" suffix, then the opening was made through a submodel.

EXAMPLES

```
! mrds_call set_modes no_list
! mrds_call open model update submodel retrieval
! mrds_call list_dbs

Open data bases are:
1         >udd>m>jg>dr>model.db
          update
2         >udd>m>jg>dr>submodel.dsm
          retrieval
```

exclusive_retrieval, er
specifies that this is unshared opening, in the sense that all update operations are prevented against any relations in this view of the database. No scope setting is necessary with this opening mode. This mode is the equivalent of opening with a retrieval mode, and doing a set_scope_all with permits of read_attr, and prevents of modify_attr, append_tuple, and delete_tuple, on these relations. Other database openers are allowed to set read_attr scope, and do retrievals on these relations.

exclusive_update, eu
specifies that this an unshared opening, in the sense that any operation is prevented by another user against any relation in this view of the database. No scope setting is necessary with this opening mode. No other database openers are allowed to set any scope on any relation in this view of the database. This mode is the equivalent of opening with an update mode, and doing a set_scope_all with permits and prevents of read_attr, modify_attr, append_tuple, and delete_tuple, on these relations. An opening with this mode will not be allowed if any relations in the opening view already have scope set by some other opening.

NOTES

The opening index, plus path and opening mode information is displayed for each opening, after a successful open operation. This can be turned off with the mrds_call set_modes no_list feature.

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. See adjust_mrds_db for the effects of this change.

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 opening modes 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, er mode requires read_attr on some attribute in each relation in the opening view, and eu 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.

EXAMPLES

```
! secure_mrds_db model -set

The database at ">udd>m>jg>dr>model.db" has been secured.

! mrds_call open model update

Error: mrds_dsl_open error by >unb>bound_mrds_|2504 Attempt to
open secured data base from model, or through non-secure submodel.
The path ">udd>m>jg>dr>model.db" refers to a database that has
been secured, and can only be be opened via a secure submodel.

mrds_call: Attempt to open secured data base from model, or
through non-secure submodel. (From dsl_$open)


! mrds_call open submodel update

Error: mrds_dsl_open error by >unb>bound_mrds_|2747 Attempt to
open secured data base from model, or through non-secure submodel.
The submodel ">udd>m>jg>dr>submodel.dsm" refers to a database
">udd>m>jg>dr>model.db" that has been secured, but the submodel
itself is not in the databases inferior directory
"secure.submodels".

mrds_call: Attempt to open secured data base from model, or
through non-secure submodel. (From dsl_$open)


! mrds_call open model.db>secure.submodels>submodel.dsm u

Open data base is:
1       >udd>m>jg>dr>model.db>secure.submodels>submodel.dsm
        update

! mrds_call close -all

! secure_mrds_db model -reset

The database at ">udd>m>jg>dr>model.db" is not secured.
```

```
! mrds_call open model er model er submodel u

Open data bases are:
1         >udd>m>jg>dr>model.db
          exclusive_retrieval
2         >udd>m>jg>dr>model.db
          exclusive_retrieval
3         >udd>m>jg>dr>submodel.dsm
          update


! display_mrds_scope_settings

Scope settings for process:  JGray.Multics.a
           process number:  2740040441

Opening index:  1
         mode:  exclusive_retrieval

          Concurrency control version:  5
                  database model path:  >udd>m>jg>dr>model.db
                     database version:  4

          Relation                              Permits    Prevents

          sample                                  r          amd

Opening index:  2
         mode:  exclusive_retrieval

          Concurrency control version:  5
                  database model path:  >udd>m>jg>dr>model.db
                     database version:  4

          Relation                              Permits    Prevents

          sample                                  r          amd

Opening index:  3
         mode:  update

          Concurrency control version:  5
                  database model path:  >udd>m>jg>dr>model.db
                     database version:  4

                  Opened via submodel:  >udd>m>jg>dr>submodel.dsm
                     submodel version:  5

          No scope currently set for this opening.
```

3.9.2.4 SET_SCOPE

The scope modes have changed for this function from the old r-u scope modes to the new security related modes of read_attr, modify_attr, append_tuple, delete_tuple.

FUNCTION: set_scope, ss

This function is used only with shared openings obtained by using the opening modes of retrieval or update. It's purpose is to set the operations that are to be permitted to the user, and the operations that are to be simultaneously prevented for other openers of the same database. The concurrency control modes, or scopes, are set on a relation basis.

USAGE

```
mrds_call set_scope data_base_index
     relation_name_1 permit_scope_1 prevent_scope_1
     { ... relation_name_N permit_scope_N prevent_scope_N }
     { wait_seconds }
```

WHERE:

1. data_base_index
   is the opening index displayed by the open function for the desired opening of the database.

2. relation_name_I
   is the name of the relation for which the concurrency control permit and prevent scope modes are to be set.

3. permit_scope_I
   is the set of operations that the user wishes to permit himself to be allowed for this relation. See the table of scope mode abbreviations below.

4. prevent_scope_I
   is the set of operations that the user wishes to deny other openers of the same database for this relation. See the table of scope mode abbreviations below.

5. wait_seconds
   is an optional argument. This is the amount of time, in seconds, the users process will wait before failing an attempt to set scope modes that conflict with another users permit and prevent scope. The full wait time is used only if the conflict remains in effect for the entire period, otherwise scope will be granted. If this argument is not given, the wait seconds defaults to 30.

NOTES

The abbreviations to be used for the scope modes for either permits or prevents are as follows.

a (or s)	append_tuple
d	delete_tuple
m	modify_attr
n	null
r	read_attr
u	update

The permit scope is made up of a concatenation of the desired operation abbreviations. If "n" permit scope is given, then no other mode may be specified for that permit. Each of "r", "a", "m", "d", and "u" may be used only once in the same permit scope. The abbreviation "u" is the same as specifying a permit scope of "amd". All this also applies to the prevent scope.

Scope settings can be displayed by the get_scope function, or the commands display_mrds_scope_settings and display_mrds_db_status.

Scope can be deleted entirely, or in part via the delete_scope function.

Scope may be set on all relations at once using the set_scope_all function.

All scope must be deleted from all relations, before scope can again be set on any relation. This prevents possible deadlock situations amoung processes requesting concurrent access protection.

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

EXAMPLES

```
! mrds_call open two_rels update

Open data base is:
1         >udd>m>jg>dr>two_rels.db
          update

! mrds_call set_scope rel1 ru n rel2 r amd

! display_mrds_scope_setttings

Scope settings for process:  JGray.Multics.a
            process number:  2740040441

Opening index:  1
          mode:  update

          Concurrency control version:  5
                  database model path:  >udd>m>jg>dr>two_rels.db
                     database version:  4

          Relation                              Permits    Prevents

          rel1                                    ramd       n
          rel2                                    r          amd


! mrds_call delete_scope_all 1

! mrds_call set_scope 1 rel1 r n

! mrds_call set_scope 1 rel2 a n

mrds_call:  Attempt to define scope while scope is not empty.
(From dsl_$set_scope)
```

3.9.2.5 SET_SCOPE_ALL

This function has had the same changes as the set_scope function.

FUNCTION: set_scope_all, ssa

This function is used only with shared openings obtained by using the opening modes of retrieval or update. It's purpose is to set the operations that are to be permitted to the user, and the operations that are to be simultaneously prevented for other openers of the same database. The concurrency control modes, or scopes, are set on all relations at once.

USAGE

```
mrds_call set_scope_all data_base_index
     permit_scope prevent_scope { wait_seconds }
```

WHERE:

1. data_base_index
   is the opening index displayed by the open function for the desired opening of the database.

2. permit_scope
   is the set of operations that the user wishes to permit himself to be allowed for all relations. See the table of scope mode abbreviations below.

3. prevent_scope
   is the set of operations that the user wishes to deny other openers of the same database for all relations. See the table of scope mode abbreviations below.

4. wait_seconds
   is an optional argument. This is the amount of time, in seconds, the users process will wait before failing an attempt to set scope modes that conflict with another users permit and prevent scope. The full wait time is used only if the conflict remains in effect for the entire period, otherwise scope will be granted. If this argument is not given, the wait seconds defaults to 30.

NOTES

The abbreviations to be used for the scope modes for either permits or prevents are as follows.

a (or s)	append_tuple
d	delete_tuple
m	modify_attr
n	null
r	read_attr
u	update

The permit scope is made up of a concatenation of the desired operation abbreviations. If "n" permit scope is given, then no other mode may be specified for that permit. Each of "r", "a", "m", "d", and "u" may be used only once in the same permit scope. The abbreviation "u" is the same as specifying a permit scope of "amd". All this also applies to the prevent scope.

Scope settings can be display by the get_scope function, or the commands display_mrds_scope_settings and display_mrds_db_status.

Scope can be deleted entirely, or in part via the delete_scope function.

Scope may be set on an individual relation basis by using the set_scope function.

All scope must be deleted from all relations, before scope can again be set on any relation. This prevents possible deadlock situations amoung processes requesting concurrent access protection.

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

EXAMPLES

```
! mrds_call open two_rels update

Open data base is:
1         >udd>m>jg>dr>two_rels.db
          update

! mrds_call set_scope_all ra md 10

! display_mrds_scope_setttings

Scope settings for process:  JGray.Multics.a
            process number:  2740040441

Opening index:  1
         mode:  update

          Concurrency control version:  5
                  database model path:  >udd>m>jg>dr>two_rels.db
                     database version:  4

          Relation                              Permits   Prevents

          rel1                                    ra        md
          rel2                                    ra        md
```

3.10 QUIESCE_MRDS_DB

The only change here, was to restrict this command to DBA use only, so that non-DBA types could not hog a database in a malicious manner.

NAME: quiesce_mrds_db, qmdb

This DBA tool quiesces a given data base, or frees it from being quiesced, for such purposes as data base backup or other exclusive activities that require a consistent and non-active data base.

USAGE

quiesce_mrds_db path {-control_args}

WHERE:

1. database_path
   is the pathname of the data base to be quiesced or freed.

2. control_args
   may be chosen from the following:

   -quiet
   causes the data base to be quiesced. (DEFAULT)

   -free
   causes the data base to be freed from a quiesced state.

   -wait_time N, -wt N
   sets the amount of time that an attempt to quiesce waits for conflicting data base users to depart before failing (see "Notes").

NOTES

Time (N) for -wait_time is in seconds. A long wait time is needed if a display_mrds_db_status shows many users; otherwise, a short wait time will suffice. The default wait time is 900 seconds. For a simple go/no go test give a wait time of 1 second.

The control args -quiet and -free are mutually exclusive, as are -free and -wait_time.

Only the quiescing process may open a quiesced data base. The user must be a DBA to make use of this command.

EXAMPLES

```
! mrds_call open qmdb update

Open data base is:
1         >udd>m>jg>dr>qmdb.db
          update

! quiesce_mrds_db qmdb -wait_time 1

quiesce_mrds_db: The specified data base is currently busy -- try
later. Unable to complete the quiescing process on the control
segment using the database path ">udd>m>jg>dr>qmdb.db".

! mrds_call close -all

! quiesce_mrds_db qmdb

! display_mrds_db_status qmdb

            Data base path: >udd>m>jg>dr>qmdb.db
                            Data base is quiesced.
                Open users: 0

! quiesce_mrds_db qmdb -free

! display_mrds_db_status qmdb

            Data base path: >udd>m>jg>dr>qmdb.db
                Open users: 0
```

3.11 UPDATE_MRDS_DB_VERSION

The only change to this command will be in documenting the fact that this command will not be usable, if the version 4 database has been put into the secure state. This is because the action of the command requires opening the database through the non-secure model view.

4.0 REFERENCES

[1] MTB-501 : The New MRDS Security Approach, J. Gray

[2] MTB-502 : Effects of Security on the MRDS Interface, J. Gray

[3] Multics Relational Data Store Reference Manual, Order Number AW53-03

[4] Logical Inquiry and Update System Reference Manual, Order Number AZ49-02

[5] MTB-506 : Extension to the create_mrds_dsm and display_mrds_dsm Commands for MRDS Security, N. Davids

[6] MTB-504 : Changes to the MRDS dsl_ Subroutine Interface, J. Gray

[7] MTB-505 : Changes to the MRDS dmd_ Subroutine Interface, J. Gray

[8] MTB-496 : Changes in the MRDS Submodel Interface, N. Davids