To: Distribution

From: N.S.Davids

Date: April 2, 1981

Subject: Extensions to the create_mrds_dsm and display_mrds_dsm commands for MRDS security

Send comments by one of the following means:

By Multics mail (on System M)
Davids.Multics (nsd.m)

By Telephone:
HVN 341-7790 or 602-249-7790

By Continuum (method of choice):
Link to transaction 450 (subject ext_cmdsm_dmdsm) in the mrds_sec meeting.

INTRODUCTION

In order for MRDS to implement attribute level security a way had to be found to define access privileges and force opening users to go through an access filter. This was done by defining access privileges in a MRDS submodel and requiring that a secure database be opened only through a submodel which is located under the secure.submodels directory which is under the database directory

This MTB describes the extensions to the create_mrds_dsm (cmdsm) command and the cmdsm source text syntax in order to implement the MRDS access filter. The changes to the display_mrds_dsm (dmdsm) command are also described. At the end of the MTB are copies of the new cmdsm and dmdsm documentation.

CHANGES TO THE SUBMODEL CREATION SCENARIO

There are two types of changes to the submodel creation scenario, changes in the cmdsm command and changes to the source text which cmdsm processes.

Changes to the cmdsm command:

The control arguments -no_list (-nls), -install (-ins), -no_install (-nins), -force (-fc), and -no_force (-nfc) have been added to the command.

The control argument -no_list suppress the production of a listing and is used to undo the effects of the -list control argument.

The -install control argument causes the submodel to be created under the secure.submodels directory which is under the database directory. This control argument is restricted to the DBA in order to prevent security violations [1].

The -no_install control argument undoes the effects of the -install control argument.

The -force control argument will cause an existing submodel with the same name to be overwritten without querying the user to be sure that this is wanted.

The control argument -no_force will turn off the effects of a -force, in other words if a submodel with the same name already exists the user will be queried before it is overwritten.

In addition to the added control arguments cmdsm was modified to check the security state of the database and the authorization of the user calling cmdsm. To prevent security violations only a DBA can run cmdsm against a secure database. The listing produced by -list now has the same format as the display produced by dmdsm when run using the -long control argument. If a DBA runs cmdsm against a secure database but does not use the -install control argument, the submodel will be created in his/her working directory and a warning will be issued stating that the created submodel is not secure. Finally a bug that forced users to use the -list control argument in order to have the submodel validated has been fixed, model relation and attribute names will always be checked for existance.

Changes in the cmdsm source text:

The cmdsm source text has been augmented to allow the submodel creator to specify access privileges at the relation and/or attribute level. These access privileges are enforced when the database associated with the submodel is a secure database.

Access to the submodel is controlled by the DBA setting Multics ACLs on the submodel entry. Anyone with read ACL on the submodel and database model can open the associated database and is subject to the access privileges specified in that submodel. It is possible for a person to have access to several submodels each with different access privileges.

Access is specified by access control statements. These control statements may appear anywhere in the submodel source, even before the relations and attributes that they define access for. Only one default relation access and one default attribute access statement can appear in a cmdsm source, however there may be multiple relation access and attribute access statements as long as each statement defines access for a different relation or attribute. The abbreviations rel_acc and attr_acc may be used in place of relation access and attribute access.

Statement Name: default relation access

Examples:

```
default relation access:
                    (relation access control list);

                    or

default relation access:
                    relation access control list;
```

Purpose:

Specifies that all relations that do not have an access set by a relation access statement will have the access specified in the relation access control list. For every submodel there is an implicit default relation access statement specifying null access, this implicit statement can be overridden by an explicit statement specifying some other access.

Statement Name: default attribute access

Examples:

```
default attribute access:
                    (attribute access control list);

                    or

default attribute access:
                    attribute access control list;
```

Purpose:

Specifies that all attributes that do not have an access set by an attribute access statement or by the "with" option in a relation access statement will have the access specified in the attribute access control list. For every submodel there is an implicit default attribute access statement specifying read access, this implicit statement can be overridden by an explicit statement specifying some other access.

Statement Name: relation access

Examples:

```
relation access:
     relation_name1 (relation access control list1),
     relation_name2 (relation access control list2)
       with attribute access
                    (attribute access control list1),
                    .
                    .
                    .
     relation_nameN (relation access control listN);

                    or

relation access:
     relation_name1 (relation access control list1);
relation access:
     relation_name2 (relation access control list2)
       with attribute access
                    (attribute access control list1);
                    .
                    .
                    .
relation access:
     relation_nameN (relation access control listN);
```

Purpose:

Specifies that the relation indicated by relation_nameI is to have the access privileges specified in the relation access control listI. The "with attribute access (attribute access control list)" clause can be considered to be a default attribute access statement which is in effect only over the asscociated relation. Access specified in the "with" clause will have precedence over access specified in the default attribute statement and will be overridden by access specified in an attribute access statement, provided an attribute access statement exists.

Statement Name: attribute access

Examples:

```
attribute access:
     attribute_name1 (attribute access control list1),
     attribute_name2 in relation_name1
       (attribute access control list2),
                        .
                        .
                        .
     attribute_nameN (attribute access control list);

                       or

attribute access:
     attribute_name1 (attribute access control list1);
attribute access:
     attribute_name2 in relation_name1
       (attribute access control list2);
                        .
                        .
                        .
attribute access:
     attribute_nameN (attribute access control listN);
```

Purpose:

Specifies that the attribute indicated by attribute_nameI is to have the access privileges specified in the attribute access control listI. If the "in relation_nameI" clause is used then the attribute will have the specified access privileges only in the indicated relation, if the "in" clause is not used then the indicated attribute will have the specified access privileges in all the relations where it occurs. There may be several attribute access statements all referring to the same attribute but having different relations specified in the "in" clause.

The access control lists contain the specifications for the access privileges. These lists are made up of a series of keywords separated by commas. The keywords depend on the access to be specified and whether the list is associated with a relation or attribute.

Relation access keywords and the operations that they allow are:

append_tuple, append tuple, or a
Specifies that tuples may be stored (e.g. using dsl_$store) in the relation.

delete_tuple, delete tuple, or d
Specifies that tuples may be deleted (e.g. using dsl_$delete) from the relation.

null, or n
Specifies that tuples may neither be stored into or deleted from the relation.

Note that any form of the access keywords may be used in the access control list. Also that a null access cannot be specified with any other access and that the order of a combination of append_tuple and delete_tuple is not important. Currently there is also the restriction that append_tuple and delete_tuple can only be specified if the submodel relation contains all the attributes that are defined in the model relation, i.e. the submodel relation is a "full view" of the model. Append_tuple has the further restriction that all the key attributes must have read_attr access set.

Attribute access keywords and the operations they allow:

read_attr, read attr, or r
Specifies that the attribute value may be read (e.g. using dsl_$retrieve).

modify_attr, modify attr, or m
specifies that the attribute value may be modified (e.g. using dsl_$modify)

null, or n
Specifies that the attribute value cannot be read or modified.

Note that any form of the attribute access keywords may be used in the access control list. Also that a null access cannot be specified with any other access and that the order of a combination of read_attr and modify_attr is not important.

Relation privileges and attribute privileges (except for the append_tuple/read_attr requirement) are independent; it is possible to have modify_attr and/or read_attr privileges on the attributes in a relation to which you do not either append_tuple or delete_tuple privileges.

Example statements:

```
default relation access: delete tuple, a;
default rel_acc: (append_tuple);
default attribute access: (n);
default attr_acc: n;
relation access: REL1 (n);
rel_acc: REL1 (null) with attr_acc (r, m);
attribute access: ATTR1 (modufy_attr, read attr);
attr_acc: ATTR1 in REL1 (read_attr);
```

CHANGES TO THE display_mrds_dsm COMMAND

This command has been modified in three ways to provide for attribute level security and in order to allow for a more usable command interface.

First, the database security state and authorization of the caller is checked. If the database is in a secured state and the caller is not a DBA then information about the model relation names and model attribute names will not be displayed.

Second, the display has been extended to be able to include the access privileges specified for relations and attributes. The display has also been reformated to be more readable.

Third, the control arguments -output_file (-of), -cmdsm, -access (-acc), -no_access (-nacc), and -no_output_file (-nof) have been added to provide for more flexibility is display formats and display destinations.

The control arguments -output_file and -no_output_file have their usual meaning of directing the output to some segment or to the terminal.

The control argument -access has an effect only if used with the control arguments -brief or -rel_names (control arguments that were implemented in previous releases), it causes the access privilege specifications for relations and attributes to be displayed. The very short, one character abbreviations are used in the display. The control argument -no_access turns off the effects of the -access control argument.

The -cmdsm control argument causes the display to be in a format that can be accepted by the create_mrds_dsm command. This control argument when used with the -output_file and -relation control arguments makes it very easy to create a subset of the submodel.

DOCUMENTATION CHANGES TO create_mrds_dsm AND display_mrds_dsm

Name: create_mrds_dsm, cmdsm

This command creates a MRDS database submodel from a data submodel source segment. The path of the resulting data submodel can specified as an argument to the dsl_$open subroutine, or the mrds_call open or linus open commands instead of the path to a database directory. This command is intended for use by data base administrators (DBAs) when defining a view of the data base for a given application. The submodel created works only against the data base whose path was given in the command, not similar data bases with other pathnames.

Usage:

create_mrds_dsm source_path db_path {-control_args}

where:

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

2. db_path
   is the pathname of the data base with which the resulting data submodel is to be associated. This data base must exist.

3. control_args
   may appear anywhere on the command line and may be chosen from the following:

   -force, -fc
   specifies that an existing submodel with the same name will be overwritten without querying the caller to be sure that the old submodel can be destroyed.

   -install, -ins
   specifies that the submodel will be created in the secure.submodels directory which is under the data base directory rather than in the working_dir (see the section on data base architecture under the create_mrds_db command). The use of this control argument will cause a directory named

secure.submodels to be created under the database directory if it does not already exist. This control argument is restricted to DBAs (see the appendix on security).

-list, -ls
specifies that a segment containing a listing of the submodel source, followed by information about the submodel to model mapping will be created in the working directory. The segment will also contain a list of any errors found while creating the submodel.

-no_force, -nfc
specifies that if a submodel with the same name already exists a query will be issued before it is overwritten. This control argument undoes the effects of a -force. (default)

-no_install, -nins
specifies that the submodel is to be created in the working_dir (default).

-no_list, -nls
specifies that a listing segment will not be created. (default)

Notes:

The data submodel is a multisegment file with the same name as the submodel source but with a dsm (rather than cmdsm) suffix.

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

Only a DBA can run this command against a secure database. If the database is secure and the -install control argument is not used the submodel will be created in the DBA's working directory and a warning that the submodel is not secure will be issued.

DATA SUBMODEL SOURCE:

The function of a data submodel is two fold, first to map the user's view of the data base into the actual data base description (i.e., the data model), second to specify relation and attribute access privileges.

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

The basic format of the create_mrds_dsm source is:

```
relation:
     relation definition 1,
                .
                .
                .
     relation definition N;

attribute access:
     attribute access definition 1,
                .
                .
                .
     attribute access definition N;

relation access:
     relation access definition 1,
                .
                .
                .
     relation access definition N;

default relation access: (relation access control list);

default attribute access: (attribute access control list);
```

Note that all of the access specification statements are optional, that multiple relation, attribute access, and relation access statements may occur and that there is no fixed order that the statements must occur in.

Relation Statement:

The relation statement(s) specifies a mapping of attributes from the data model relation to the data submodel relation. This mapping can be used to change the names of the data model relations and attributes, to reorder the attributes within a relation, to omit attributes from a relation and to omit relations from the database view. Multiple relation statements can occur as long as they all define different data submodel relation to data model relation mappings.

Examples:

```
relation:
     relation1 (attribute1 ... attributeN),
     relation2 = model_relationI (attribute1 ... attributeN),
                              .
                              .
                              .
     relation3 (attribute1 ... attributeI = model_attributeK
                                  ... attributeN);

                              or

relation:
     relation1 (attribute1 ... attributeN);
relation:
     relation2 = model_relationI (attribute1 ... attributeN);
relation:
     relation3 (attribute1 ... attributeI = model_attributeK
                                  ... attributeN);
```

If the data submodel relation name differs from that specified in the data model, the data submodel relation name is equated to the corresponding name in the data model. If only one relation name is supplied in the data submodel relation expression, it is assumed that the data submodel and data model relation names are the same. A data submodel relation name may be up to 64 characters long, it may be composed of letters, numbers, hyphens, and underscores but must begin with a letter.

Similarly, if the data submodel view of an attribute name differs from that in the data model, the data submodel attribute name is equated to the corresponding name in the data model. If only one name for an attribute is supplied, it is assumed that the data submodel and data model names for the attribute are the same. A data submodel attribute name may be up to 64 characters long, it may be composed of letters, numbers, hyphens, and underscores but must begin with a letter.

Access Specification Statements:

The cmdsm source text has been augmented to allow the submodel creator to specify access privileges at the relation and/or attribute level. These access privileges are enforced when the database associated with the submodel is a secure database.

Access to the submodel is controlled by the DBA setting Multics ACLs on the submodel entry. Anyone with read ACL on the submodel and the database model can open the associated database and is subject to the access privileges specified in that submodel. It is possible for a person to have access to several submodels each with different access privileges.

Access is specified by access control statements. These control statements may appear anywhere in the submodel source, even before the relations and attributes that they define access for. Only one default relation access and one default attribute access statement can appear in a cmdsm source, however there may be multiple relation access and attribute access statements as long as each statement defines access for a different relation or attribute. The abbreviations rel_acc and attr_acc may be used in place of relation acces and attribute access.

Statement Name: default relation access

Examples:

```
default relation access:
                    (relation access control list);

                    or

default relation access:
                    relation access control list;
```

Purpose:

Specifies that all relations that do not have an access set by a relation access statement will have the access specified in the relation access control list. For every submodel there is an implicit default relation access statement specifying null access, this implicit statement can be overridden by an explicit statement specifying some other access.

Statement Name: default attribute access

Examples:

default attribute access:
(attribute access control list);

or

default attribute access:
attribute access control list;

Purpose:

Specifies that all attributes that do not have an access set by an attribute access statement or by the "with" option in a relation access statement will have the access specified in the attribute access control list. For every submodel there is an implicit default attribute access statement specifying read access, this implicit statement can be overridden by an explicit statement specifying some other access.

Statement Name: relation access

Examples:

```
     relation access:
          relation_name1 (relation access control list1),
          relation_name2 (relation access control list2)
            with attribute access (attribute access control list1),
                          .
                          .
                          .
          relation_nameN (relation access control listN);

                          or

     relation access:
          relation_name1 (relation access control list1);
     relation access:
          relation_name2 (relation access control list2)
            with attribute access (attribute access control list1);
                          .
                          .
                          .
     relation access:
          relation_nameN (relation access control listN);
```

Purpose:

Specifies that the relation indicated by relation_nameI is to have the access privileges specified in the relation access control listI. The "with attribute access (attribute access control list)" clause can be considered to be a default attribute access statement which is in effect only over the asscociated relation. Access specified in the "with" clause will have precedence over access specified in the default attribute statement and will be overridden by access specified in an attribute access statement, provided an attribute access statement exists.

Statement Name: attribute access

Examples:

```
     attribute access:
          attribute_name1 (attribute access control list1),
          attribute_name2 in relation_name1
            (attribute access control list2),
                          .
                          .
                          .
          attributeNameN (attribute access control list);

                          or

     attribute access:
          attribute_name1 (attribute access control list1);
     attribute access:
          attribute_name2 in relation_name1
            (attribute access control list2);
                          .
                          .
                          .
     attribute access:
          attributeNameN (attribute access control listN);
```

Purpose:

Specifies that the attribute indicated by attribute_nameI is to have the access privileges specified in the attribute access control listI. If the "in relation_nameI" clause is used then the attribute will have the specified access privileges only in the indicated relation, if the "in" clause is not used then the indicated attribute will have the specified access privileges in all the relations where it occurs. There may be several attribute access statements all referring to the same attribute but having different relations specified in the "in" clause.

The access control lists contain the specifications for the access privileges. These lists are made up of a series of keywords separated by commas. The keywords depend on the access to be specified and whether the list is associated with a relation or attribute.

Relation access keywords and the operations that they allow are:

append_tuple, append tuple, or a
Specifies that tuples may be stored (e.g. using dsl_$store) in the relation.

delete_tuple, delete tuple, or d
Specifies that tuples may be deleted (e.g. using dsl_$delete) from the relation.

null, or n
Specifies that tuples may neither be stored into or deleted from the relation.

Note that any form of the access keywords may be used in the access control list. Also that a null access cannot be specified with any other access and that the order of a combination of append_tuple and delete_tuple is not important. Currently there is also the restriction that append_tuple and delete_tuple can only be specified if the submodel relation contains all the attributes that are defined in the model relation, i.e. the submodel relation is a "full view" of the model. Append_tuple has the further restriction that all the key attributes must have read_attr access set.

Attribute access keywords and the operations they allow:

read_attr, read attr, or r
Specifies that the attribute value may be read (e.g. using dsl_$retrieve).

modify_attr, modify attr, or m
Specifies that the attribute value may be modified (e.g. using dsl_$modify).

null, or n
Specifies that the attribute value cannot be read or modified.

Note that any form of the attribute access keywords may be used in the access control list. Also that a null access cannot be specified with any other access and that the order of a combination of read_attr and modify_attr is not important

Relation privileges and attribute privileges (except for the append_tuple/read_attr requirement) are independent; it is possible to have modify_attr and/or read_attr privileges on the attributes in a relation to which you do not either append_tuple or delete_tuple privileges.

EXAMPLES:

The following examples show different submodels which are all defined over the the states database which is described in the examples of the create_mrds_db command. The first submodel is a view full submodel, i.e. all the relations in the model are present and each relation has all the attributes that were defined in the model.

cmdsm_source_example_1

```
/*
    This submodel is a simple view corresponding to the
    entire database with no name changes. Since no access is
    specified the default relation access of null and the
    the default attribute access of read is used.
*/

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);
```

cmdsm_source_example_2

```
/*
    This submodel renames the last_name and first_name
    attributes to ln and fn and omits the salary attribute
    from the person relation. The attribute key has been
    moved to the first position in the person_state relation
    which has also been renamed to ps. The relation
    state_location has been omitted from this submodel.
*/

relation:
          person (ln = last_name fn = first_name expenses),
          state_history (key same_name date_time text),
          ps = person_state (key last_name first_name);
```

cmdsm_source_example_3

```
/*
    This submodel specifies a default relation access of
    append_tuple and delete_tuple and a default attribute
    access of read_attr and modify_attr.  Notice that
    comments can be placed between both relations and
    attributes.
*/

default relation access: (append_tuple, delete tuple);

default attribute access: (modify attr, read attr);

relation:
          person                /* person relation */
              (ln = last_name   /* last name of person */
               fn = first_name  /* first name of person */
               expenses         /* expenses of person to date */
              ),

          person_state          /* location of person */
              (last_name        /* same as ln in person */
               first_name       /* same as fn in person */
               key              /* state key */
              );
```

cmdsm_source_example_4

```
/*
    This submodel specifies a default relation access of
    append tuple and delete tuple and a default attribute
    access of read attr and modify attr. Access for the
    person relation is set to append with a default
    attribute access of read attr. Note that all access key
    words and the statement keywords are in their short form
    and the multiple use of the relation, relation access
    and attribute access statements.

    A display of the submodel with the relation and
    attribute access may be found in the examples for the
    display_mrds_dsm command.
*/

default rel_acc: a, d;

default attr_acc: r, m;

attr_acc:
          salary (r),
          last_name (r),
          first_name (r);

rel_acc:
          state_history (a),
          person_state (d),


relation:
          person (last_name first_name salary expenses),
rel_acc:
          person (a) with attr_acc (r);
attr_acc:
          salary in person (n);


relation:
          person_state (last_name first_name key);


relation:
          state_history (key state_name date_time text);
attr_acc:
          key in state_history (r);
```

The following examples show command usage:

An invocation of the command using no control arguments, this is the same as invoking the command with control arguments of -no_list, -no_force, and -no_install.

```
create_mrds_dsm cmdsm_source_example_1.cmdsm states.db
```

This invocation will create the submodel in the secure.submodels directory under the states.db directory. Only a DBA can use this control argument. The command line has been split into two lines only to make it fit onto the page.

```
create_mrds_dsm     -install     cmdsm_source_example_2.cmdsm
     states.db
```

The following invocation will both install the submodel in the secure.submodels directory and write over any existing cmdsm_source_example_3 without querying the invoker. The command line has been split into two lines only to make it fit onto the page.

```
create_mrds_dsm -force cmdsm_source_example_2.cmdsm -install
     states.db
```

This last example installs the submodel in the secure.submodels directory, forces the overwritting of an existing submodel with the same name, and produces a listing called cmdsm_source_example_2.list in the working directory. Notice that the short form of the control arguments and the command name are used.

```
cmdsm -fc cmdsm_source_example_2 states -ls -install
```

Name: display_mrds_dsm, dmdsm

This command displays information about the specified MRDS data submodel.

Usage:

display_mrds_dsm dsm_path {-control_args}

where:

1. dsm_path
   is the pathname of the data submodel file to be displayed. If dsm_path does not have a suffix of dsm, then one is assumed. However the dsm suffix must be the last component of the name of the data submodel file.

2. control_args
   may be chosen from the following:

-access, -acc
   specifies that access information (both relation and attribute) is to be displayed. This control argument has effect only with the -rel_names and -brief control arguments. (default)

-brief, -bf
   specifies that only the submodel relation names and attribute names are to be displayed. This control argument may be superseded by any of -cmdsm, -rel_names, or -long which follow it in the command line. (default)

-cmdsm
   specifies that the display is to have a format that may be processed by the create_mrds_dsm command to produce another submodel. This control argument is limited to DBAs if the submodel is associated with a secure database. This control argument may be superseded by any of -long, -rel_names, or -brief which follow it in the command line.

-long, -lg
specifies that the display is to contain all the information that is in the submodel. This includes the database path, submodel version, submodel creation date and creator, submodel relation names and associated model relation names, submodel attribute names and associated model attribute names, relation and attribute access, and the attribute data types. If the person running this command is not a DBA and the submodel is associated with a secure database then the model relation names and model attribute names will not be displayed. This control argument may be superseded by any of -cmdsm, -rel_names, or -brief which follow it in the command line.

-no_access, -nacc
specifies that access information is not to be displayed. This control argument has effect only with the -rel_names and -brief control arguments.

-no_output_file, -nof
causes the output display to be written to the terminal. This control argument will undo the effects of the -output_file control argument (default).

-output_file path, -of path
causes the output display to be written to the specified path instead of to the terminal. Anything already stored in the segment at path will be overwritten.

-rel_names, -rn
specifies that only submodel relation names are to be displayed. This control argument may be superseded by any of -cmdsm, -brief, or -long which follow it in the command line.

-relation REL_1 REL_2 ... REL_N
specifies that information about REL_1 through REL_N is to be displayed. The information about each relation is displayed in the order they are specified. If some specified relation REL_I does not exist in the submodel an error is reported and the display proceeds with the next relation. If the display is going to an output file the error is reported both to the terminal and the output file. This control argument may be used with the control arguments -cmdsm, -long, -rel_names and -brief to produce a display of part of the submodel (default is to display all relations).

EXAMPLES:

The following examples all use the submodel example_4 which was generated from the create_mrds_dsm example cmdsm_source_example_4. The submodel secure_example_4 is the same submodel defined for a secure database.

```
! display_mrds_dsm example_4

        person                  a
          last_name             r
          first_name            r
          salary                n
          expenses              r

        person_state            d
          last_name             r
          first_name            r
          key                   rm

        state_history           a
          key                   r
          state_name            rm
          date_time             rm
          text                  rm
```

```
! display_mrds_dsm example_4 -long

Submodel path:       >udd>Multics>examples>example_4
      Version:       5
   Created by:       Davids.Multics.a
   Created on:       03/10/81 1059.6

Database path:       >udd>Multics>examples>states.db
      Version:       4
   Created by:       Davids.Multics.a
   Created on:       03/10/81 1130.3


Submodel Relation Name:        person
           Model Name:         person
               Access:         append_tuple

          Submodel Attribute Name:      last_name
                     Model Name:        last_name
                         Access:        read_attr
                      Data Type:        char (32)
                                        Indexed

          Submodel Attribute Name:      first_name
                     Model Name:        first_name
                         Access:        read_attr
                      Data Type:        char (32)

          Submodel Attribute Name:      salary
                     Model Name:        salary
                         Access:        null
                      Data Type:        fixed dec (59, 2) unal

          Submodel Attribute Name:      expenses
                     Model Name:        expenses
                         Access:        read_attr
                      Data Type:        fixed dec (59, 2) unal


Submodel Relation Name:        person_state
           Model Name:         person_state
               Access:         delete_tuple

          Submodel Attribute Name:      last_name
                     Model Name:        last_name
                         Access:        read_attr
                      Data Type:        char (32)
                                        Indexed
```

```
          Submodel Attribute Name:      first_name
                      Model Name:       first_name
                          Access:       read_attr
                       Data Type:       char (32)

          Submodel Attribute Name:      key
                      Model Name:       key
                          Access:       read_attr modify_attr
                       Data Type:       bit (70)

Submodel Relation Name:         state_history
           Model Name:          state_history
               Access:          append_tuple

          Submodel Attribute Name:      key
                      Model Name:       key
                          Access:       read_attr
                       data Type:       bit (70)
                                        Indexed

          Submodel Attribute Name:      state_name
                      Model Name:       state_name
                          Access:       read_attr modify_attr
                       Data Type:       char (30)
                                        Indexed

          Submodel Attribute Name:      date_time
                      Model Name:       date_time
                          Access:       read_attr modify_attr
                       Data Type:       fixed bin (71)

          Submodel Attribute Name:      text
                      Model Name:       text
                          Access:       read_attr modify_attr
                       Data Type:       char (4096) var
```

```
! display_mrds_dsm example_4 -cmdsm

/*
created from: >udd>Multics>examples>example_4.dsm
         for:  >udd>Multics>examples>states.db
          by:    display_mrds_dsm -cmdsm
*/


relation access:    person (append_tuple);

relation:           person = person
                             (last_name = last_name
                              first_name = first_name
                              salary = salary
                              expenses = expenses);

attribute access:   last_name in person (read_attr),
                    first_name in person (read_attr),
                    salary in person (null),
                    expenses in person (read_attr);

/* ************************************************************ */

relation access:    person_state (delete_tuple);

relation:           person_state = person_state
                                 (last_name = last_name
                                  first_name = first_name
                                  key = key);

attribute access:   last_name in person_state (read_attr),
                    first_name in person_state (read_attr),
                    key in person_state (read_attr modify_attr);

/* ************************************************************ */

relation access:    state_history (append_tuple);

relation:           state_history = state_history
                                  (key = key
                                   state_name = state_name
                                   date_time = date_time
                                   text = text);

attribute access:   key in state_history (read_attr),
                    state_name in state_history (read_attr modify_attr),
                    date_time in state_history (read_attr modify_attr),
                    text in state_history (read_attr modify_attr);
```

```
! display_mrds_dsm example_4 -relation_names

          person
          person_state
          state_history
```

```
! display_mrds_dsm example_4 -relation person_state -long

Submodel path:      >udd>Multics>examples>example_4
      Version:      5
   Created by:      Davids.Multics.a
   Created on:      03/10/81 1059.6

Database path:      >udd>Multics>examples>states.db
      Version:      4
   Created by:      Davids.Multics.a
   Created on:      03/10/81 1130.3


Submodel Relation Name:       person_state
            Model Name:       person_state
                Access:       append_tuple

          Submodel Attribute Name:      last_name
                       Model Name:      last_name
                           Access:      read_attr
                        Data Type:      char (32)
                                        Indexed

          Submodel Attribute Name:      first_name
                       Model Name:      first_name
                           Access:      read_attr
                        Data Type:      char (32)

          Submodel Attribute Name:      key
                       Model Name:      key
                           Access:      read_attr modify_attr
                        Data Type:      bit (70)
```

display_mrds_dsm

```
! display_mrds_dsm example_4 -relation state_history person -no_access

state_history
          key
          state_name
          date_time
          text

person
          last_name
          first_name
          salary
          expenses
```

```
! display_mrds_dsm secure_example_4 -relation person_state -long

Submodel path:       >udd>Multics>examples>secure_example_4
      Version:       5
   Created by:       Davids.Multics.a
   Created on:       03/10/81 1059.6

Database path:       >udd>Multics>examples>states.db
      Version:       4
   Created by:       Davids.Multics.a
   Created on:       03/10/81 1130.3


Submodel Relation Name:        person_state
               Access:         append_tuple

          Submodel Attribute Name:      last_name
                           Access:      read_attr
                        Data Type:      char (32)
                                        Indexed

          Submodel Attribute Name:      first_name
                           Access:      read_attr
                        Data Type:      char (32)

          Submodel Attribute Name:      key
                           Access:      read_attr modify_attr
                        Data Type:      bit (70)
```

REFERENCES

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