HONEYWELL

MULTICS RELATIONAL DATA STORE REFERENCE MANUAL

SOFTWARE

MULTICS RELATIONAL DATA STORE REFERENCE MANUAL

SUBJECT

Description of the Multics Data Base Manager (Multics Relational Data Store)

SPECIAL INSTRUCTIONS

This manual supersedes AW53, Revision 3, dated June 1980 and Addendum A dated October 1980. The manual has been extensively revised. Several appendixes have been reorganized into sections. Change bars in the margin denote technical additions and changes; asterisks denote deletions. Sections 7, 11, 13 and Appendix F are completely new and do not contain change bars.

Refer to the Preface for additional MR9.0 information.

SOFTWARE SUPPORTED

Multics Software Release 9.0

ORDER NUMBER

AW53-04

September 1981

Honeywell

PREFACE

This manual is a combined data base primer and reference manual for the Multics Relational Data Store (MRDS). It describes the functions and subroutine interfaces to a relational type of data base organization. This manual is intended for users familiar with the general characteristics of Multics, including the environment of an interactive terminal session, and assumes the user has a basic understanding of the simpler features of the PL/I language, since all examples are written in PL/I.

This manual contains references to the Multics Commands and Active Functions, Order No. AG92, referred to as Commands, the Multics Subroutines and Input/Output Modules, Order No. AG93, referred to as Subroutines, and the Multics Programmer's Reference Manual, Order No. AG91, referred to as Reference Manual.

This manual contains descriptions of the following Multics Priced Separate products (PSPs); some of them may not be installed in your system.

(SGD6801) LINUS (Logical Inquiry and Update System)
(SGL6805) MRPG (Report Generator) Facility
(SGU6801) SORT/MERGE Facility

Significant Changes in this Addendum

Revised the display_mode_dm command (refer to Section 3):

- Changed -attribute and -domain control arguments
- Added -crossref control argument

Deleted all data from Section 11.

The following changes were made to the restructure_mrds_db command (refer to Section 14):

- Added three new control arguments (-force, -no_force, and -relation_type)

File No.: 1L13

12/86
AW53-04D

- Added seven new restructure requests (create_attribute, create_domains, delete_attribute, delete_domain, rename_attribute rename_domain, and rename_relation)

- Expanded the display_data_model restructure request

- Added three new control arguments to the ready_db restructure request (-force, -no_force, and -relation_type).

CONTENTS

SECTION 1

INTRODUCTION

The relational data base interface, known as the Multics Relational Data Store (MRDS), provides Multics users with a general data base management facility that is callable from Multics command level and from programming languages that support the standard Multics call interface. A full range of data base definition, retrieval, and update capabilities is available, together with facilities that provide a large measure of data independence and control of concurrent accesses to the data base.

BASIC DATA BASE CONCEPTS

data base (DB)
: an integrated collection of operational data (i.e., data that can be read, written, or modified).

data base manager (DBM)
: a software system designed to make an integrated collection of data available to a variety of users while providing security measures to ensure privacy where desired.

data base administrator (DBA)
: the person responsible for defining and creating the data base and for controlling its use.

user
: a person, subject to administrative controls, who retrieves, updates, or deletes data within the data base. Anyone having access to the Multics system can be both the administrator and user of his own data base. In general, however, a user is one of many others who access a common data base which they do not administer.

data independence
: a characteristic of a data base management system that allows the user to be more concerned with the information content and logical properties of the data base and less concerned with the data's physical organization and location. A high degree of data independence implies that the syntax of inquiries to the data base manager is relatively insensitive to changes in the physical organization of the data base.

data model (DM) or schema
: the total description of a data base, defining the characteristics and organization of all the data within the data base. This description allows users to reference data in logical rather than physical terms. The data base administrator creates the data model; the data base manager references the data model prior to accessing the associated data base.

data submodel (DSM) or subschema
: an alternate, and usually incomplete, description of an existing data base that may optionally be provided to users of the data base. This alternate description enhances "data independence" by allowing users to concern themselves with only a particular subset of an

existing data base and/or reference a data base with alternate (alias) names. The data submodel may be created by either the user or the data base administrator. The user may then "open" the submodel, creating the illusion of accessing the data base defined in the submodel instead of the actual data base.

DATA BASE TERMINOLOGY

The relational and network approaches to data base management are based on differing philosophies and each approach has its own terminology. The relational approach draws upon terms found in the precise mathematical theory of relations (attributes, tuples, etc.), whereas the network approach draws upon terms common to the data processing world (field, record, etc.). As illustrated in the following data base and table example, there exists nearly a one-to-one correspondence between the two sets of terminology.

PRESIDENTS DATABASE

NAME	PARTY	HOME-STATE
Eisenhower	Republican	Kansas
Kennedy	Democrat	Massachusetts
Johnson	Democrat	Texas
Nixon	Republican	California

COMPARISON OF TERMINOLOGY

DEFINITION	ILLUSTRATION	RELATIONAL TERMINOLOGY	NETWORK TERMINOLOGY
A file. A collection of organized data.	The set of data given above.	relation	record type
A record. A representative "row" of data.	"Kennedy,Democrat, Massachusetts"	tuple	record
The name of a data field within a record; a column of information	"HOME-STATE"	attribute	field or data-item
The value of a data field within a record.	"Texas"	attribute value	value of field or data-item
The set of all values a data field may assume.	The names of the 50 states.	domain	not used
User's definition of the data base.	Created by user & administrator.	data submodel	subschema
Total definition of the data base.	Created by the administrator.	data model	schema
Associated terms:		MRDS	CODASYL, I-D-S/II

CHARACTERISTICS OF MRDS

The data management system can be called from programming languages supporting the standard Multics call interface as well as from Multics command level via the mrds_call command.

All data bases reside within the Multics storage system as directories, segments, and files, and are protected by the security features inherent to the Multics virtual memory environment.

MRDS uses relational data base structures that are based on the mathematical theory of relations.

Inquiries to MRDS consist of a single logical request containing a selection_expression that defines the goal of the search through the data base. For example:

```
"-range (x Person)
 -select x.emp_num
 -where x.name = ""Smith"" "
```

This expression (defined in detail in Section 2) contains a relation (file) named "Person". The expression defines the subset of employee numbers in the Person relation that are assigned to employees named Smith. This set of employee numbers may be compiled, deleted, modified, or retrieved depending on the user's intention.

SECTION 2

USERS' GUIDE

This section, which is a primer for the MRDS interface, contains general explanations for several commands and subroutines described in detail in Sections 3 through 5.

The primer closely follows the actual sequence of events in a typical session with MRDS. Although the exact command invocations and examples shown may be duplicated for training purposes by a terminal user, they should not be interpreted as representing a rigid or necessary sequence of operations. Rather, each example outlines the general function and typical usage of a command or subroutine. The user should examine the detailed descriptions provided in later sections to build examples that fit a particular requirement.

In this section the use of any particular command or subroutine is not an exhaustive description of its capabilities. This section uses only those features that are essential or instructive for the novice user of MRDS. The Commands, Subroutines, and Subsystem Writers' Guide sections of this document contain a description of the more advanced features of MRDS.

In all examples, the longest and most descriptive name of a command or subroutine is used for clarity. Examples are often stylistically formatted for aesthetic reasons only. Such formatting should not be construed as mandatory, or even recommended, since MRDS accepts all commands and source text in free form.

This section contains information for both the data base administrator and the data base user. Readers who are only interested in accessing an established data base need not concern themselves with:

- MRDS Tutorial (steps 1 and 2)
- Data Base Design

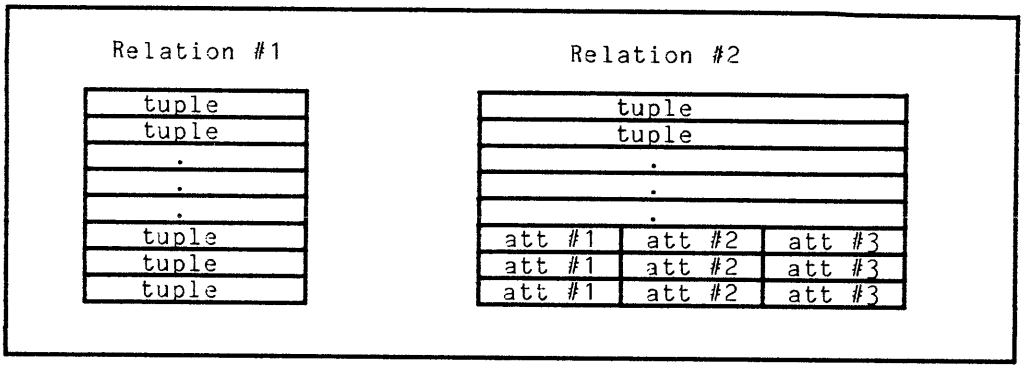
It is assumed that the reader is familiar with the basic concepts and terminology contained in Section 1.

BASIC MRDS CONCEPTS

A relational data base is best viewed as a simple tabular or columnar arrangement of data divided into one or more groups called relations. The data within a given data base has been placed there because it, in some sense, fits together and is used or collected for some common purpose. The data within a relation of the data base can be considered as data belonging to some subclass or subset of the overall data base.

A relational data base is most easily pictured as a series of columns that form a table:

DATA BASE "A"



where:

1. A relational data base contains at least one relation (file).

2. Each relation contains at least one tuple (record); otherwise, it is considered an unpopulated relation.

3. An unpopulated data base contains only unpopulated relations.

4. A tuple contains at least one attribute (field).

5. All tuples within a given relation have the same format.

6. Some fixed set of one or more attributes in each relation must uniquely identify each tuple in that relation. (These attributes combined are called the primary key of the relation.)

MRDS TERMINOLOGY

access
: the ability to perform any combination of data base operations.

compile
: converts a selection expression to internal structure format and saves it for the life of a data base opening.

data base
: a Multics directory containing the model (schema) definition of the data, data storage files, and access control structures.

DBA
: a data base administrator, defined as someone holding "sma" ACL on the data base directory.

delete
: deletes a complete tuple from a relation.

exclusive
: a qualifier to an opening mode that prohibits concurrent updating by other users.

model
: the main view (schema) defining the data base and its data.

modify
: alters one or more attributes of a tuple existing within a relation (excluding the attributes which make up the primary key).

open
: readies a data base for user access.

populated
: a data base in which at least one of its relations contains at least one tuple of data; otherwise, the data base is unpopulated.

primary key
: the set of attributes (one or more) whose values are used to uniquely identify a tuple in a relation.

retrieval
: an opening mode for a data base that allows only retrieval operations.

retrieve
: returns some data subset of the data base.

scope
: how the user intends to share the data base with others.

secured
: a data base that has had the command secure_mrds_db with the set control argument run against it and has not subsequently had the secure_mrds_db command with the -reset control argument run against it.

  A secured submodel is located under the secure.submodels directory, which is under the data base directory, for the purpose of providing attribute level access controls.

selection expression
: the specification of the relations referenced, attributes selected, and conditions required to uniquely identify the desired tuples.

shared
: the qualifier to an opening mode that allows concurrent access by other users. Unless an exclusive mode is specified (e.g., exclusive update), data bases are opened in a shared mode.

store
: adds a complete tuple to a relation.

submodel
: a structure providing an alternate view (subschema) of the main model view (schema) of the data base data. It also contains the relation and attribute access specification used when the data base is secured.

tuple
: an instance of data (a record) stored in a relation made up of individual attributes (fields).

unpopulated
: a relation containing no tuples or a data base containing only unpopulated relations.

update
: an opening mode for a data base that allows all data base operations.

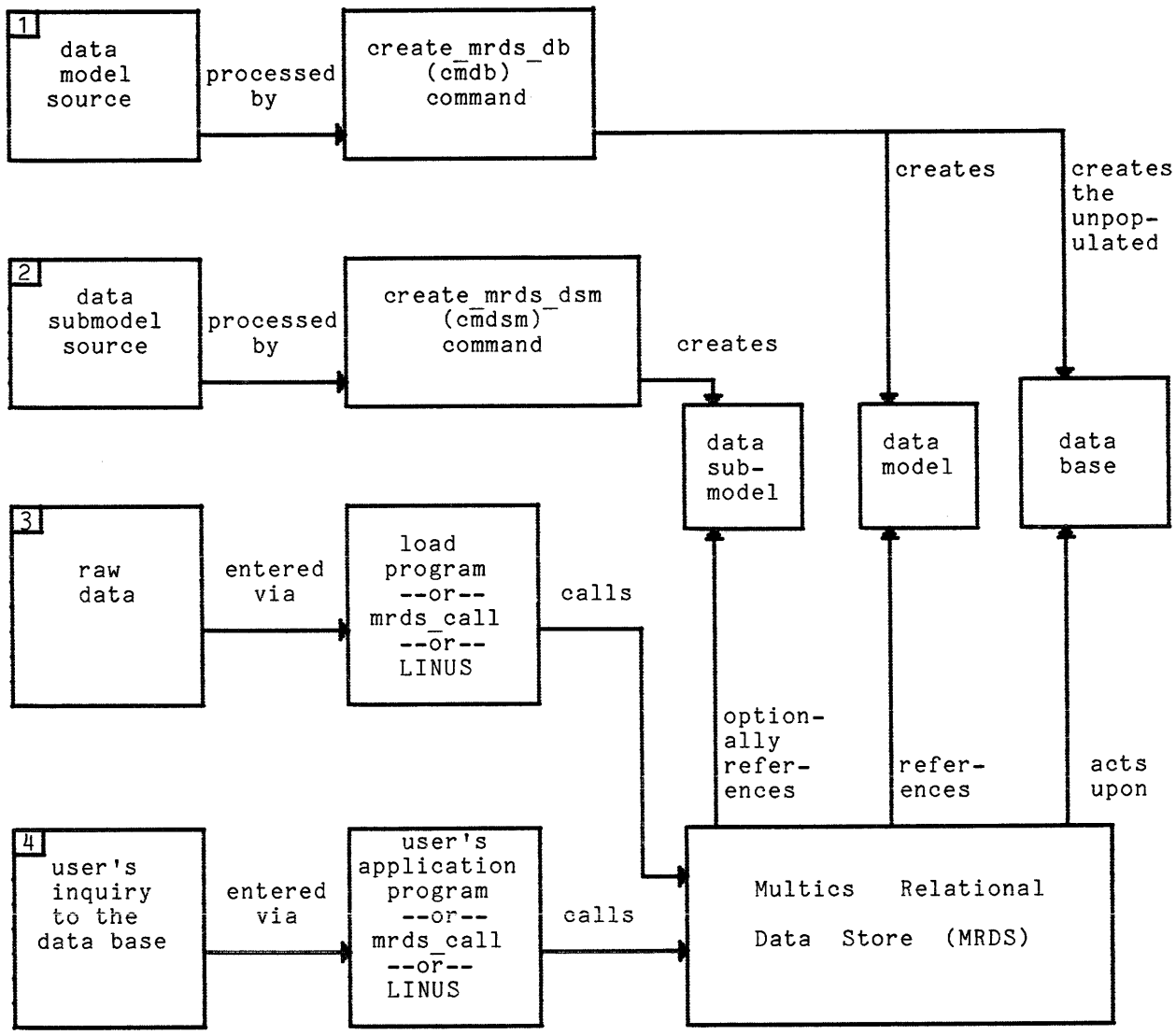
view
: the logical relation and attribute makeup of the data base provided by a model or submodel. A view may be a subset of the entire data base.

FUNCTIONAL DIAGRAM

The process of creating and accessing a data base consists of four basic steps:

1. Create the data model and the corresponding unpopulated data base.
2. Create an optional data submodel.
3. Load the unpopulated data base.
4. Access the populated data base.

The following diagram illustrates these four steps.



The numbers (1,2,3, and 4) on the diagram correspond to the numbering used in this tutorial. Steps 1 and 2 describe processes generally done once by the data base administrator. Steps 3 and 4 concern users accessing an established data base. Users who are only interested in accessing an established data base need not concern themselves with steps 1 and 2.

The examples used in this section are PL/I examples, many of which utilize the Data SubLanguage (DSL) subroutines. (Parallel mrds_call examples are included under the mrds_call command description). In addition, the Logical Inquiry and Update System (LINUS) may also be used to access an established data base.

1. Creating the data model and the corresponding unpopulated data base.

   a. The data base administrator decides to create a relational data base.

   b. Using one of the Multics text editors, the administrator builds a text segment called the data model source. This segment contains a description of the data and its organization within the desired data base. For example, let the segment "foo.cmdb" contain the following text:

```
domain:      char_12   char(12),
             char_5    char(5);

attribute:   name      char_12,
             emp_num   char_5,
             comp      char_5;

relation:    Employee (name  emp_num*  comp),
             Comp_mgr (comp* emp_num);
```

   This example defines a data base consisting of two relations (Employee and Comp_mgr). The domain and attribute statements define the names and characteristics of three attributes[1] (name, emp_num, and comp) and the relation statement defines the names and composition of the two relations.

   The domain statement is not a structure declaration and the order of the attribute names has no significance here. The domain statement is simply a list of names and associated data types. (The definable data types are a subset of the Multics PL/I data types.) In this example, the "emp_num" attribute is defined to be a 5-character string.

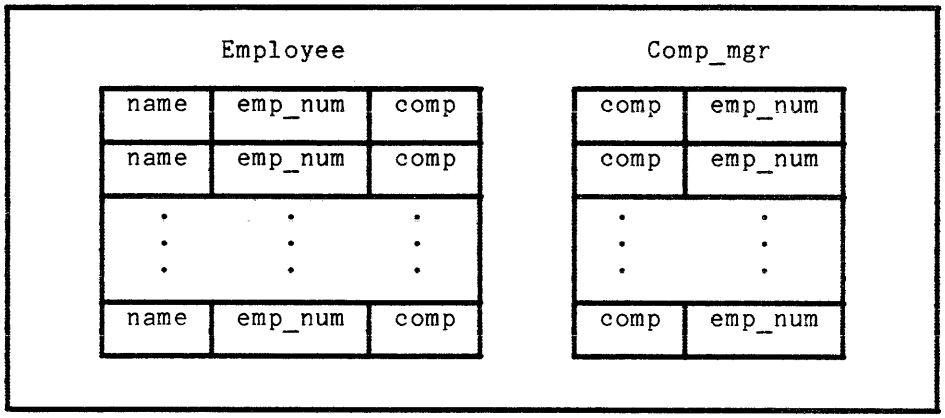
   The relation statement determines the logical structure of the data base. It is here that the number of relations, the names of the relations, and the logical composition of the relations are defined. The Employee relation (record) is defined to consist of three attributes (fields): name, emp_num, and comp in that order.

   The asterisks in the relation statements designate the attributes that are the primary keys of the relations. The primary keys in this example are emp_num and comp. A primary key must be designated and it must be unique. Since no two employees can have the same employee number, "emp_num" is a good choice for the primary key of the Employee relation. Additional information regarding the meaning and selection of primary keys is provided under "Primary and Secondary Indexes" described later in this section.

---

[1] The domain statement actually defines the characteristics of the set of domains. It also defines a set of identically-named attributes having those characteristics. The attribute statement is used to define attribute names for use in relations over generic domain data types. This is explained in more detail under the heading "Domains and Attributes" later in this section.

The following diagram illustrates the data base defined in the above data model source:

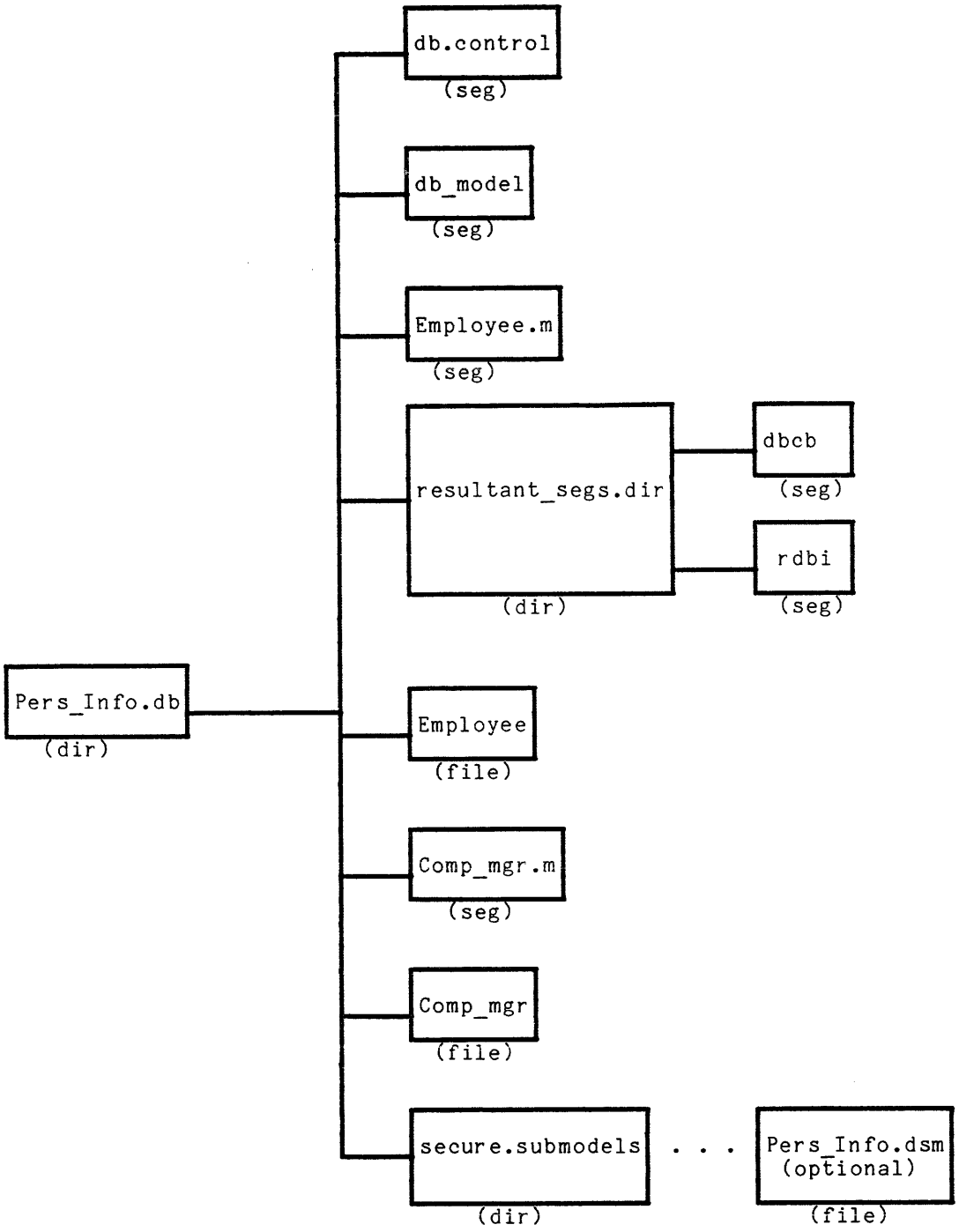


The Employee relation is a list of employee names, employee numbers, and the name of the component each employee is assigned to. The Comp_mgr relation is a list of all components in the firm and the employee number of the manager of each component.

c. The administrator now invokes the create_mrds_db (cmdb) command in order to translate the data model source and create the desired data base.

```
create_mrds_db foo.cmdb Pers_Info.db
```

Using the description found in the segment "foo.cmdb", this command creates a data base within the user's working directory consisting of a directory named Pers_Info.db and all subordinate segments, files, and directories required to implement the defined data base. For the current example, the data base has the following file structure.



Notice that three directories, six segments, and three files are required to implement the Pers_Info.db data base. The data base control segment, "db.control", is used internally by MRDS to control concurrent access to the data base. The db_model segment holds common data base information, such as descriptions of domains and a list of relation names.

The segments ending in ".m" hold the model information for the corresponding relations. The files having the relation names hold the actual data and are managed by various file managers. The "secure.submodels" directory holds submodels that provide attribute level security. The Pers_Info.dsm file is an optional submodel, which can be created as shown below.

Knowledge of the file structure of the data base, while informative, is normally of no concern to the data base user since the MRDS interface makes the structure transparent.

The resultant_segs.dir contains a copy of the internal structures used by MRDS on the open data base. The internal structures are in the dbcb and rdbi segments under this directory. This copy makes the opening of the data base faster. It is created when the data base is created via the cmdb command. If an existing data base is opened by a DBA, the copy is created only if it does not already exist. Note that MRDS can open the data base even if this directory is absent, but it takes a bit more time.

The DBA can use the Multics system ACL commands (see MPM Commands manual) to set the appropriate access controls to the files (relations), segments, and the containing directory. For example, if a user is only to retrieve information from the Employee relation, but is allowed to modify information within the Comp_mgr relation, the administrator must give that user the following minimum access rights:

```
db.control          rw  (read and write)
db_model            r   (read)
Employee.m          r   (read)
Comp_mgr.m          r   (read)
Employee            r   (read)
Comp_mgr            rw  (read and write)
dbcb                r   (read)
rdbi                r   (read)
resultant_segs.dir  s   (status)
```

See Section 7 for more details on security.

2. Creating an optional data submodel.

   a. A submodel is an alternate description of an existing data base. When a user opens a data base using the name of an associated submodel instead of the name of the actual data base, the user's view of the data base corresponds to the data base described in the submodel. This gives the user the illusion of accessing the data base defined by the submodel. A submodel could be used when:

      (1) It is desirable for some users to have a simpler subset view of a large data base.

      (2) It is desirable to reference the data base and its contents using alternate or alias names instead of using the names actually defined in the data model.

      (3) It is desirable to make the restructuring of a data base transparent to the data base users. For example, application programs that reference a particular relation containing four attributes need not be rewritten if a fifth attribute is added to the relation in a redesign of the data base. The programs need only "open" a submodel that defines that relation as having the original four attributes.

      (4) Attribute level security is to be provided (see Section 7).

   b. A data submodel may be created at any time by either a user or the data base administrator by creating a text segment called a data submodel source. This segment, like the data model source, contains a description of the desired data base. However, unlike the data model source, the data base relations and attributes described here must be a subset and/or a renaming or reordering of an existing data base's relations and attributes. In addition, the data submodel source may only contain relation statements and no domain or attribute statements. The number of attributes present in these relation statements must be less than or equal to the actual number of attributes in the existing relations. Alias names, if desired, are defined by setting the new name equal to (=) the actual name used in the existing data base. For example, let the segment named Pers_Info.*.cmdsm contain the following text:

```
relation: Employee (last_name = name emp_num),
                   comp = Comp_mgr (comp emp_num);
```

      The ordering of terms around the equal sign is significant and must be <desired name> = <actual name>.

      Notice that comp is the name of both a relation and an attribute in this example. This is allowed. Notice also that no asterisks are used in the syntax of submodels.

This text describes an alternate view of the Pers_Info.db data base which is logically equivalent to the following data model source:

```
domain:     char_12   char(12),
            char_5    char(5);

attribute:  last_name char_12,
            emp_num   char_5,
            comp      char_5;

relation:   Employee (last_name  emp_num*),
            comp (comp*  emp_num);
```

c. The create_mrds_dsm (cmdsm) command is now invoked as in the following example:

```
create_mrds_dsm Pers_Info >udd ...  >Pers_Info.db -install
```

Using the description found in Pers_Info.cmdsm, this command creates a data submodel named Pers_Info.dsm under the secure.submodels directory of the data base and associates the submodel with the Pers_Info.db data base apparently located in a directory other than the user's working directory. Opening the data base with the pathname for Pers_Info.dsm instead of Pers_Info.db creates the illusion that the user is accessing the data base defined in 2b above instead of the actual data base defined in 1b. The number of submodels associated with a given data base is arbitrary.

d. Two restrictions exist when using a data submodel that defines a relation as being a subset of the actual relation in the data base (i.e., the submodel relation contains fewer attributes than the actual relation and thus is a partial view of the relation). The operations that are restricted are storing tuples in or deleting tuples from such a relation when using the submodel.

3. Loading the unpopulated data base.

a. Depending on the form and quantity of the data, the administrator may elect to input that data using a terminal, a tuple at a time, using the command interface mrds_call or the LINUS store request, or to write and execute a load program designed to read the raw data from existing file(s) and store it into the data base using calls to the dsl_$store subroutine. In addition, the LINUS store request may be used to load relations from raw text files if the format of the files is identical to the format of the relations.

b. Prior to loading, the data base must be opened in a manner similar to opening a file for I/O. One of the following four opening modes must be given:

Open Mode	Open Mode Constant Name	Value
retrieval	retrieval	1
update	update	2
exclusive retrieval	exclusive_retrieval	3
exclusive update	exclusive_update	4

The named constants are defined in the "mrds_opening_modes_.incl.pl1" system include file.

The following PL/I example illustrates a subroutine call that opens the data base Pers_Info.db in an exclusive update mode (declarations are omitted).

```
call dsl_$open ("Pers_Info.db", dbi_1, exclusive_update, code);
```

If the opening is successful, MRDS assigns and returns an integer, called the data base index (dbi), which remains unique to the data base during the current user session (from opening to closing). A user process may have 128 data bases open at one time and must refer to each opened data base by the assigned data base index in all subsequent calls.

c. The storing of data into a data base may occur at any time during the life of a data base and may in fact be a continuing process. Quite often, a data base is created as a container for data previously stored in another media. In this case, the initial transfer of data into the data base is a process known as loading (populating) the data base and is actually no more than a series of store operations.

This example illustrates the loading of the data base Pers_Info.db using data found in two existing data files named emp_data and comp_data. (Not all PL/I declarations are shown, but the relation structure declarations may be obtained by using the create_mrds_dm_include command.)

emp_data

Akins	57111	Eng
Hamilton	48227	Mfg
Morton	48350	Eng
Shaw	51603	Mfg
Whiting	49189	Fin
Nielson	52464	Eng

comp_data

Mfg	51603
Eng	48350

```
declare 01  Employee,
            02  name char(12),
            02  emp_num char(6),
            02  comp char(6);
declare 01  Comp_mgr,
            02  comp char(6),
            02  emp_num char(6);
read file (emp_data) into (Employee);
call dsl_$store (dbi_1, "Employee", Employee.name,
            Employee.emp_num, Employee.comp, code);
do while (^eof_emp_data);
        read file (emp_data) into (Employee);
        call dsl_$store (dbi_1, "-another", Employee.name,
             Employee.emp_num, Employee.comp, code);
end;
read file (comp_data) into (Comp_mgr);
call dsl_$store (dbi_1, "Comp_mgr", Comp_mgr.comp,
     Comp_mgr.emp_num, code);
do while (^eof_comp_data);
        read file (comp_data) into (Comp_mgr);
        call dsl_$store (dbi_1, "-another", Comp_mgr.comp,
             Comp_mgr.emp_num, code);
end;
```

NOTE: Data conversion is performed automatically by MRDS and proceeds according to the standard PL/I conversion rules. In this example the char(6) comp of the Employee structure is converted into the char(5) comp attribute in the data base. This data conversion is only available for assign_ data types, which excludes the picture data type.

If an incomplete tuple is being stored (i.e., a tuple with one or more unknown attribute values), the user must insert null values into the unknown attribute of the tuple being stored in order to prevent a shifting of attribute values into the wrong attribute location. One rule used in this case is to substitute a " " (blank) for attributes requiring alphabetic data and a -1 (or some type of numeric value that cannot be confused with valid data) for an attribute requiring numeric data.

The Pers_Info.db data base is considered populated when the load program completes its execution. Its logical appearance is:

Pers_Info.db

Employee

name	emp_num*	comp
Hamilton	48227	Mfg
Morton	48350	Eng
Whiting	49189	Fin
Shaw	51603	Mfg
Nielson	52464	Eng
Akins	57111	Eng

Comp_mgr

comp*	emp_num
Eng	48350
Mfg	51603

The actual internal order of the data within the data base and the corresponding order in which the data might be retrieved are functions of internal implementation and should not be anticipated by the user. Standard Multics sort commands and subroutines are

available for users desiring sorted data (refer to the Multics SORT/MERGE manual). LINUS will also return sorted data if desired.

d. After loading, if no further accessing is desired, the data base is closed.

```
call dsl_$close (dbi_1, code);
```

4. Accessing the populated data base.

a. In order to access an established data base, a user must first open the data base in the desired open mode. Every user must individually open a data base even though other users may have the same data base currently open.

```
call dsl_$open ("Pers_Info", dbi_1, update, code);
```

This example opens the data base Pers_Info.db in the update mode. If another user has the data base currently open in an exclusive mode, MRDS returns a code indicating that the data base is busy. Otherwise, a unique data base index is returned and the user may proceed.

An array illustrating those times when a user requesting a data base opening may receive a "busy" code is shown below. For example, a user requesting an opening mode of exclusive retrieve (er) of a data base currently open in an update (u) mode with an update form of scope set (e.g., for store), receives a busy code and must wait until the other user closes the data base.

Open Request \ Another User's Current Open Mode	r	u	er	eu
r				busy *
u			busy *	busy *
er		busy *		busy %
eu	busy *	busy *	busy %	busy %

(*) The r or u shared modes will only conflict when conflicting scope has been or is requested.

(%) The er and eu modes will only conflict if the opening views contain the same relations.

b. If the data base is opened in one of the shared modes (i.e., update or retrieval), MRDS prevents access operations until the user performs the set_scope operation. Set_scope is the process of declaring how the data base is to be shared with other users by specifying:

(1) The operations the user intends to perform on the relations included in the user's view of the data base (called permit_ops)

(2) The operations others are to be prevented from performing on the relations in their view of the data base (called prevent_ops)

Permit_ops and prevent_ops are initiated by the dsl_$set_scope subroutine call by specifying the sum of scope mode encoding(s) that correspond to the desired data base operation(s).

Scope Code	Operation
0	null
1	read_attr or read
2	append_tuple or store
4	delete_tuple or delete
8	modify_attr or modify

```
call dsl_$set_scope (dbi_1, "Employee", 1, 15,
   "Comp_mgr", 3, 14, 120, code);
```

In this example, the user is indicating:

(a) To permit the retrieve operation (permit_ops = 1) in the Employee relation and retrieve and store operations (permit_ops = 1 + 2) in the Comp_mgr relation for himself.

(b) Other users are prevented from performing any operation on the Employee relation (prevent_ops = 1 + 2 + 4 + 8) and are prevented from performing store, delete, or modify operations on the Comp_mgr relation (prevent_ops = 2 + 4 + 8), while retrieve operations may still be done by others on this relation. (See "Additional Capabilities", described later in this section, on dynamically changing scope.)

(c) The length of time (in this case 120 seconds) the user is willing to wait for the scope request to be satisfied. If another user has the data base currently open with a scope that conflicts with this request, MRDS automatically queues the request with the intent of satisfying it when the conflicting scope is relinquished. If the specified wait time is exceeded without the scope request being satisfied, MRDS returns a code indicating that the data base is busy. Otherwise the user may proceed.

The user may specify the number of seconds to wait to satisfy the set scope request (there is no anticipated maximum), or may elect to use the MRDS default value of 30 seconds by omitting the argument.

c. Once the data base is open (and, if a shared opening, the set_scope request is accepted) the user can access the data base in any manner desired, subject of course to access restrictions imposed by the data base administrator and to self-imposed opening mode and scope restrictions.

Access Mechanisms Other Than Store

To access an open data base, the user must make a subroutine call to the appropriate MRDS entry point in order to:

- Supply the name of the operation to be performed (i.e., delete, modify, or retrieve).

- Supply the data base index of the target data base.
- Specify that subset of the data within the data base upon which the operation is to be performed.

The name of the operation is indicated by the dsl_ entry name used in the subroutine call. The three possibilities are:

```
call dsl_$delete ( ... );

call dsl_$modify ( ... );

call dsl_$retrieve ( ... );
```

The data base index (dbi) is always supplied as the first argument in the subroutine call. For example:

```
call dsl_$delete (dbi, ... );
```

The subset of the data within the data base to be operated upon is defined by the second argument known as a selection expression (see "Formal Definition of the Selection Expression" in Section 4). A selection expression is a character-string argument normally containing a range clause, a select clause, and a where clause, each identified respectively by the keywords -range, -select and -where. For example:

```
call dsl_$modify (dbi,
    "-range ...
     -select ...
     -where ... ", ... );
```

The construction of a selection expression is best understood after the user has mentally performed the desired operation on a tabular representation of the data base. During this process, the user identifies:

- The relations to be referenced in order to accomplish the desired operation (range clause).
- The attributes affected, that is, the attributes to be retrieved, modified, or deleted (select clause).
- The conditions required to uniquely identify the desired tuples (where clause).

The relations to be referenced by these data base operations appear in the -range clause of the selection expression, the attributes to be affected appear in the -select clause, and the conditions required to identify the desired tuples are specified in the -where clause.

In order to illustrate the construction of a selection expression, assume that the employee tuples of all Engineering employees having an employee number greater than 50000 are to be deleted from the Employee relation of the Pers_Info data base above. Note the following:

- The Employee relation is the only relation that needs to be accessed in order to accomplish the objective.
- Since only entire tuples can be deleted, all of the attributes of the Employee relation are affected (name, emp_num, and comp).
- There are only two conditions required to select the desired tuples. The employee number must be greater than 50000 and the component must be equal to "Eng".

The following PL/I subroutine call accomplishes the desired deletions. The multiline formatting is shown only for the sake of clarity. PL/I declarations are not shown.

```
call dsl_$delete (dbi_1,
     "-range (E Employee)
      -select E.name E.emp_num E.comp
      -where ((E.emp_num >""50000"") &
              (E.comp = ""Eng""))", code);
```

The -range clause is said to assign a "tuple variable" E to the Employee relation. Tuple variables may be given any name, but generally a one- or two-character abbreviation for the designated relation is chosen. Tuple variables should be thought of as pointers which are moved about the selected relation by MRDS while it attempts to satisfy the conditions specified in the -where clause. It is sometimes necessary to assign two or more tuple variables to the same relation.

The -where clause specifies the conditions necessary to identify the tuples of interest. In this case, the employee number must be greater than 50000 and the component must be equal to the string "Eng". The double quoting is required in order to resolve the ambiguity of quotes within quotes. The parentheses are required for efficient parsing of the selection expression.

The -select clause lists the attributes to be affected when a tuple is found that satisfies the conditions specified in the -where clause. For this data base, the tuples containing the names of Nielson and Akins satisfies the specified conditions and cause their deletion. For the convenience of the user, a tuple variable appearing by itself in a -select clause is interpreted to mean all attributes, allowing the above subroutine call to be rewritten as:

```
call dsl_$delete (dbi,
     "-range (E Employee )
      -select E
      -where ((E.emp_num > ""50000"") &
              (E.comp = ""Eng""))", code);
```

MRDS assumes no precedence for the boolean operators "&" and "|"; therefore, parentheses must direct a specific order of evaluation. Selection expressions may be arbitrarily complex and may include the following operators:

Algebraic Operators	Boolean Operators
= (equal to)	& (and)
< (less than)	| (inclusive or)
> (greater than)	^ (not)
<= (less than or equal to)	
>= (greater than or equal to)	
^= (not equal to)	

Selection expressions can be compiled to reduce the overhead of translation. Compiled selection expressions can be used directly in the execution of define_temp_rels, retrieves, modifies, and deletes. The following PL/I subroutine call compiles a desired selection expression.

```
call dsl_$compile (dbi,
     "-range (e employee)
      -select e.name e.emp_num
      -where (e.comp = ""SW-ENGR"")", se-index, code);
```

Compiled selection expressions are freed or released by calling the compile entrypoint in the normal manner and supplying a negative number for the selection expression index of the compiled selection expression that is to be deleted. In this case, the contents of the selection expression are unimportant. For example:

```
dsl_$compile(db_index, "", -se_index, code);
```

Note: Compiled selection expressions cannot be retained from one data base opening to another. It is not necessary to explicitly release a compiled selection expression when the application is finished with it; however, it is good practice. When an open data base is closed, any compiled selection expressions that exist at the time are released automatically.

EXAMPLE 1

Hamilton has transferred to the Engineering component. Modify his component name to read "Eng":

```
call dsl_$modify (dbi_1,
   "-range (E Employee)
    -select E.comp
    -where E.emp_num = ""48227"" ", "Eng", code);
```

A selection expression is used to uniquely define the subset of the data base to be modified. The attribute values selected are sequentially replaced with the values provided to the right of the selection expression. In this example, only one attribute is selected (E.comp) and, therefore, only one replacement value is provided (Eng).

NOTE: Modification of a primary key attribute (i.e., an attribute followed by an asterisk in the data model source) is not allowed and results in a code indicating an invalid operation. Such modifications may only be accomplished by deleting the entire tuple and storing a new tuple containing the corrected values. (The reason for this is explained later, under "Primary and Secondary Indexes".) Modify operations are further restricted to include only attributes contained in the same relation (i.e., in order to modify both Employee.comp and Comp_mgr.comp, two modify operations are required).

EXAMPLE 2

Retrieve the Employee tuple of employee "Shaw":

```
call dsl_$retrieve (dbi_1,
   "-range (E Employee)
    -select E
    -where E.name = ""Shaw"" ",
     arg_1, arg_2, arg_3, code);
```

The above selection expression identifies the one Employee tuple having the name attribute equal to Shaw. The retrieve operation returns the three attribute values: Shaw, 51603, and Mfg. When performing retrieve operations, users must supply the correct number of arguments to hold the returned attribute values. Data conversion (if it occurs) proceeds according to the standard PL/I conversion rules.

EXAMPLE 3

Retrieve the Employee tuples of all manufacturing and finance employees:

```
call dsl_$retrieve (dbi_1,
   "-range (E Employee)
    -select E
    -where (E.comp = ""Mfg"") ¦ (E.comp = ""Fin"")",
     arg_1, arg_2, arg_3, code);

do while (code = 0);
    put skip list (arg_1, arg_2, arg_3);
    call dsl_$retrieve (dbi_1,
       "-another", arg_1, arg_2, arg_3, code);
end;
```

This PL/I example illustrates the typical programming construct required when retrieving more than one set of attributes from a data base. The first call to dsl_$retrieve sets up the selection conditions and returns one set of attributes satisfying the -where clause. The second call to dsl_$retrieve requests another set of attributes satisfying the same selection conditions specified in the first call, by using "-another" as the selection expression. The "code" returned is zero if the retrieve is successful. If no tuple satisfied the selection condition then the code returned is mrds_error_$tuple_not_found.

The above example returns the three tuples (nine attributes) currently in the Employee relation where either "Mfg" or "Fin" is the comp. If the -where clause in this example were eliminated, the Employee tuples of all employees (not just the "Mfg" and "Fin" employees) would be retrieved.

It should be noted that the retrieve operation call is satisfied and complete when the first tuple that matches the selection expression is found. Therefore, as in the example above, additional calls using the "-another" selection expression must be made to find subsequent tuples. The delete and modify operations, on the other hand, require only one call to operate on all matching tuples in the data base.

EXAMPLE 4

Retrieve the employee number of Hamilton's manager:

```
call dsl_$retrieve (dbi_1,
   "-range (E Employee) (C Comp_mgr)
    -select C.emp_num
    -where ((E.name = ""Hamilton"") &
     (E.comp = C.comp))", arg_1, code);
```

This example returns the employee number 51603. Notice the usage of two tuple variables in the range clause and the selection of only one attribute value from the Comp_mgr tuple. (Terms like "E.comp = C.comp" and "C.comp = E.comp" are identical in meaning and may be used with either ordering.)

EXAMPLE 5

Retrieve the name and employee number of Hamilton's manager:

```
call dsl_$retrieve (dbi_1,
   "-range (E1 Employee) (C Comp_mgr) (E2 Employee)
    -select E2.name E2.emp_num
    -where ((E1.name = ""Hamilton"") &
     ((E1.comp = C.comp) & (C.emp_num = E2.emp_num)))",
     arg_1, arg_2, code);
```

This example returns the two attribute values Shaw and 51603. The select clause could also have used C.emp_num instead of E2.emp_num.

EXAMPLE 6

Delete the Employee tuple of all employees whose last name begins with the letters M through Z:

```
call dsl_$delete (dbi_1,
   "-range (E Employee)
    -select E
    -where ([substr(E.name 1 1)] >= ""M""
     & ([substr(E.name 1 1)] <= ""Z"")", code);
```

This example illustrates the substr (substring) built-in function and identifies a one character-long substring starting with the first character of E.name. The square brackets are required to designate built-in functions or expressions within a selection expression.

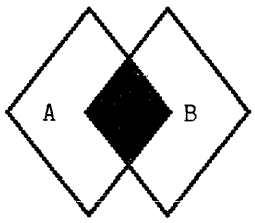
EXAMPLE 7

The following example illustrates the format for using the MRDS set operators. (Refer to Appendix D for information regarding set operators.)

```
call dsl_$retrieve (dbi_1,
   "(-range ...
     -select ...
     -where ...)

  -inter
    (-range ...
     -select ...
     -where ...)", arg_1, ... , code);
```

selection expression 1 -- selects all of A in the following diagram.

the intersection (-inter) of selection expression 1 and 2 -- selects the shaded area of A and B in diagram (elements that belong to both A and B).



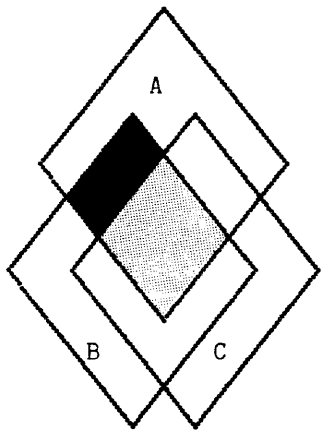
Several selection expressions (up to 20) may be strung together, separated by set operators, in order to construct one complex selection expression where each set operation is applied to the previous selection expression or result of a previous set operation, such as:

```
call dsl_$retrieve (dbi_1,
     "((-range  ...                    } see a below
        -select ...
        -where  ...)                   }see b below
-inter
      (-range  ...
       -select ...
       -where  ...))                                  } see c below
-differ
      (-range  ...
       -select ...
       -where  ...)  ", arg_1, ... , code);
```

a. Selection expression 1 -- selects all of A in the following diagram.

b. The intersection (-inter) of selection expression 1 and 2 -- selects all of the shaded area of A and B in the diagram (elements that belong to both A and B).

c. The difference (-differ) of selection expression 3 and the result of the previous operations (selection expressions 1 and 2) -- select only the dark shaded area of B in the diagram (elements that belong to B but do not belong to C).



The attributes selected in each selection expression must be identical in number and must also be domain-compatible (see "Domains and Attributes" below). The set operators may only be used in retrievals and defining temporary relations. They may not be used to delete or modify data.

EXAMPLE 8

The following example illustrates one of the useful features of a compiled selection expression using the combination of -compile and -another to obtain data from two different MRDS data bases. (Refer to "Argument Substitution Using .V. and .X." below for additional information on use of the ".X." in the example.)

```
call dsl_$compile (dbi_2, "-range (s sale s)
                           -select s.item
                           -where (s.name = .X.)", se_index, code);

call dsl_$retrieve (dbi_1, "-range (e emp) " ||
                           "-select e.name", emp_name, code_1);

do while (code_1 = 0);

   put skip list (emp_name);

   put skip list;

   call dsl_$retrieve (dbi_2, "-compiled", se_index, emp_name, item, code_2);

   do while (code_2 = 0);

      put skip list (item);

      call dsl_$retrieve (dbi_2, "-another", emp_name, item, code_2);

   end;

   call dsl_$retrieve (dbi_1, "-another", emp_name, code_1);

end;

call dsl_$compile (dbi_2, "", -se_index, code_1;
```

The first call to dsl_$compile translates the supplied selection expression into disjunctive normal form, stores it, and returns the selection index which is used to indicate that the selection expression is compiled. The se_index returned from dsl_$compile is used in the dsl_$retrieve with "-compiled" as the selection expression.

The last call to dsl_$compile deletes the compiled select expression by utilizing the negative selection expression index.

Three powerful concepts complement the capabilities of MRDS:

- Scope deletion
- Temporary relations
- Argument substitution using ".V." and ".X."

Scope Deletion

The delete scope request lets a user remove all or part of a previously set scope. Recall that setting scope is required when opening a data base in a shared mode (i.e., update or retrieval). If a user's scope includes permit_ops or prevent_ops that conflict with the scope of others attempting to use the data base, those users are placed in a queue to wait until the current user either closes the data base, deletes sufficient scope, or the allowed waiting time limit is exceeded. Deleting scope can, therefore, be considered an act of courtesy to others awaiting access to a shared data base and users should be alert to the possibility of relaxing their scope restrictions whenever completing a logical phase of their session with the data base.

```
call dsl_$set_scope (dbi_1, Employee, 15, 15, code);
call dsl_$dl_scope (dbi_1, Employee, 14, 1, code);
.
.
.
.
```

Prior to the execution of the delete_scope request in the above example, other users are prevented from performing retrieval and update operations in the Employee relation. After executing the delete_scope request, other users are only prevented from performing updates in the Employee relation, including the fact that the user has revoked his own update permission. A subsequent attempt by the user to update an Employee tuple would result in a code indicating a scope violation.

Although a user may repeatedly set and delete scope while the data base is open, the user must delete all scope before setting a new scope. This rule avoids potential deadlock problems within the data base manager.

Users of heavily shared data bases should cooperate to maximize the availability of those data bases. Combinations of opening modes and scope restrictions, in order of concurrency, are:

1. Exclusive_update mode.
2. Exclusive_retrieval mode.
3. Update or retrieval mode with many permit_ops and many prevent_ops.
4. Update or retrieval mode with few permit_ops and few prevent_ops.

Temporary Relations

A temporary relation is a user-created subset of an open data base that is accessible via selection expressions in the same manner as permanent relations for retrieve operations only. Temporary relations are created for the purpose of simplifying selection expressions, reducing the access time to otherwise dispersed data, or obtaining a count of the tuples retrieved by a selection expression. Temporary relations reside in the user's temporary storage directory determined by set_mrds_temp_dir with a default of the process directory and, although temporary relations and permanent relations are physically and logically identical, a temporary relation is destroyed whenever it is redefined, deleted, or whenever the associated data base is closed.

The asterisks used in a define_temp_rel request designate the attributes (or concatenation of attributes) that are to be used as the primary key of the temporary relation. (Temporary relations cannot have secondary keys.) Users must exercise care when selecting primary keys for temporary relations since MRDS automatically (and without warning) removes duplicate key tuples from the resulting relation.

```
rel_index_1 = 0;
call dsl_$define_temp_rel (dbi_1,
   "-range (x Employee) (y Comp_mgr) (z Employee)
    -select x.emp_num* z.name
    -where  ((x.comp = y.comp) &
        (y.mgr_emp_num = z.emp_num))", rel_index_1, code);
```

This example creates (because rel_index_1 = 0) a temporary relation containing the names of all employees and their respective managers as represented in the table below. Notice that this information did not exist side-by-side in the original data base. Unlike the retrieve request that returns one set of selected attributes for each call, the define_temp_rel request selects all attributes that satisfy the selection expression and physically places them into the temporary relation.

temporary relation 1

emp_num*	name
48227	Shaw
48350	Morton
51603	Shaw
52464	Morton
57111	Morton

The name Whiting does not appear in the example because the sample data base does not include a manager of "Fin" in the Comp_mgr relation. Therefore the -where selection expression cannot be satisfied for Whiting's Employee tuple.

Users may define, redefine, or delete any number of temporary relations. However, no more than 20 per data base opening may exist at any one time. For each temporary relation created, MRDS assigns and returns an integer, called a relation index (rel_index), which remains unique to the temporary relation and its associated data base during the current user session. A user must reference an existing temporary relation by using the assigned relation index in the range clause of a retrieve selection expression (refer to "Argument Substitution Using .V." described below).

The accessing of temporary relations is restricted to retrieve and define_temp_rel operations only. The delete, store, and modify operations are not allowed for temporary relations.

```
rel_index_1 = 1;
call dsl_$define_temp_rel (dbi_1,
   "-range (x Employee) (y Comp_mgr) (z Employee)
    -select x.emp_num z.name*
    -where  ((x.comp = y.comp) &
        (y.mgr_emp_num = z.emp_num))", rel_index_1, code);
```

This example illustrates a redefinition of a temporary relation. If the relation index variable has a value of zero when the define_temp_rel request is made, the resulting temporary relation is assigned the next available relation index. If however the relation index is greater than zero on input and if a temporary relation possessing this index already exists within the data base, that temporary relation is redefined. In this case, the old temporary relation "1" has been replaced with a new temporary relation "1". If the specified relation index is less than zero and a temporary relation exists whose index is equal to the absolute value of the index given, then that temporary relation is deleted.

The primary key attribute has been changed to z.name, a non-unique attribute. Consequently, the new temporary relation "1" contains less information than the old temporary relation "1".

temporary relation 1

emp_num	name*
48350	Morton
51603	Shaw

The number of tuples in a temporary relation can be determined using the dsl_$get_population interface.

Argument Substitution Using ".V." and ".X."

Since a MRDS selection expression is passed as a character-string argument, some mechanism is needed that allows a programmer to insert variable values into a selection expression when the application program is executing. Consider the case where the variable emp_num_1 contains a previously selected employee number and the programmer wishes to use this employee number in a MRDS selection expression.

The method of substituting a value into a selection expression involves the MRDS argument ".V.". This argument may only be used in place of a relation name in the range clause (for temporary relation indexes only, not relation names) and in place of a constant in the where clause. When present in a selection expression, MRDS sequentially replaces the .V. argument(s) with the value(s) of the variable(s) of literal(s) immediately following the selection expression.

```
call dsl_$retrieve (dbi_1,
   "-range (x Employee)
    -select x.name
    -where (x.emp_num = .V.)",
     emp_num_1, arg_1, code);
```

The .V. argument is also the mechanism used to insert temporary relation indexes into the range clause of a selection expression. When the .V. argument is used within the range clause, then the relation to which it refers can only be a temporary relation. Assume that the variable "temp_rel_1" contains the index of an existing temporary relation. The following example is identical to the one above except for the use of the temporary relation instead of the Employee relation:

```
call dsl_$retrieve (dbi_1,
   "-range (x .V.)
    -select x.name
    -where (x.emp_num = .V.)",
     temp_rel_1, emp_num_1, arg_1, code ;
```

The ".X." argument is a substitution character similar to ".V.". It can only be used when compiling a selection expression and it is used to specify an argument that is not known at the time of compilation.

```
call dsl_$compile (db_index,
                   "-range (x Employee)
                    -select x.name x.emp_num x.comp
                    -where ((x.emp_num < .V.) &
                            (x.comp > .X.)),
                   se_index, {.V._values, ... ,} code)
```

The .V._values is required to satisfy any .V. argument substitution characters that exist in the selection expression. These then become constant (i.e., they cannot be changed in a later reference to the selection expression). This can be done using the .X. substitution character in the selection expression at compilation time. The .V. values are not needed in future references to the compiled selection expression. No values are necessary at this time to satisfy any of the .X.s that may exist in the where clause. They are only used in the define_temp_rel, retrieve, modify, or delete procedures. If se.index is equal to zero on input, a new index is returned in se_index. If se_index is equal to some positive value and the index is currently being used (i.e., already assigned to a compiled selection expression), that compiled selection expression is redefined.

NOTE: .X. values are not allowed in expressions and function references. Two examples of its use are:

```
... "-where [substr (a.value .X. 2)] = ""31"" "

... "-where a.value = 2 * .X."
```

DATA BASE DESIGN

The design of a data base is a responsibility of the data base administrator. It requires insight into the nature and form of the data to be stored and must include an understanding of the manner in which the user community is expected to access the data base. The design chosen by the data base administrator affects such characteristics as:

1. Overall size of the data base
2. Processor time required to effect a given update request
3. Complexity of the selection expressions required to update the data base
4. Internal logical considerations called update and deletion anomalies (discussed below)

The novice designer of a relational data base often has the tendency to create a one-relation data base that contains all of the data. However, a well-designed relational data base typically contains several relations with the data distributed among them. This partitioning of a data base into several relations is characteristic of a data base design process known as normalization. Normalization is described at the end of this section, but first the reader should understand the desirability of a multi-relation data base over a one-relation data base.

Consider two data bases that contain logically equivalent information but differ in their physical structure. The Pers_Info_A data base consists of only one relation whereas Pers_Info_B (containing the same information as Pers_Info_A) is partitioned into two relations that share a common attribute (comp).

The .V. argument is also the mechanism used to insert temporary relation indexes into the range clause of a selection expression. When the .V. argument is used within the range clause, then the relation to which it refers can only be a temporary relation. Assume that the variable "temp_rel_1" contains the index of an existing temporary relation. The following example is identical to the one above except for the use of the temporary relation instead of the Employee relation:

```
call dsl_$retrieve (dbi_1,
   "-range (x .V.)
    =select x.name
    -where (x.emp_num = .V.)",
     temp_rel_1, emp_num_1, arg_1, code ;
```

DATA BASE DESIGN

The design of a data base is a responsibility of the data base administrator. It requires insight into the nature and form of the data to be stored and must include an understanding of the manner in which the user community is expected to access the data base. The design chosen by the data base administrator affects such characteristics as:

1. Overall size of the data base
2. Processor time required to effect a given update request
3. Complexity of the selection expressions required to update the data base
4. Internal logical considerations called update and deletion anomalies (discussed below)

The novice designer of a relational data base often has the tendency to create a one-relation data base that contains all of the data. However, a well-designed relational data base typically contains several relations with the data distributed among them. This partitioning of a data base into several relations is characteristic of a data base design process known as normalization. Normalization is described at the end of this section, but first the reader should understand the desirability of a multi-relation data base over a one-relation data base.

Consider two data bases that contain logically equivalent information but differ in their physical structure. The Pers_Info_A data base consists of only one relation whereas Pers_Info_B (containing the same information as Pers_Info_A) is partitioned into two relations that share a common attribute (comp).

Pers Info A

Employee

name	emp_num*	comp	mgr_num	bldg
Hamilton	48227	MFG	51603	A
Morton	48350	ENG	48350	B
Whiting	49189	FIN	49189	B
Shaw	51603	MFG	51603	A
Nielson	52464	ENG	48350	B
Akins	57111	ENG	48350	B
Green	57183	MFG	51603	A

Pers Info B

Employee

name	emp_num*	comp
Hamilton	48227	MFG
Morton	48350	ENG
Whiting	49189	FIN
Shaw	51603	MFG
Nielson	52464	ENG
Akins	57111	ENG
Green	57183	MFG

Comp_mgr

comp*	mgr_num	bldg
ENG	48350	B
MFG	51603	A
FIN	49189	B

A first observation about these data bases is the size difference between Pers_Info_A and Pers_Info_B. For a large number of employees, Pers_Info_B would contain approximately 40% fewer attributes than Pers_Info_A. A carefully chosen partitioning of a relation into multiple relations usually produces this effect.

A second observation concerns the number of changes required to reflect one "real world event" such as the engineering manager being changed from Morton to Nielson. Notice that three attributes must be changed in order to update Pers_Info_A, but only one attribute must be changed in order to update Pers_Info_B. This irregularity in Pers_Info_A is known as an update anomaly; and although the update can, in both cases, be done with only one MRDS modify request, MRDS must do considerably more work for the user requesting the update of Pers_Info_A. Counter examples exist, but in well-designed data bases update anomalies are generally minimized by partitioning large relations into smaller relations.

Now consider the case in which Whiting has retired. This change can be accomplished in both data bases by deleting Whiting's Employee tuple. However, there are side effects which may be undesirable. By deleting Whiting's Employee tuple from Pers_Info_A, the fact that the Finance component was located in building "B" has also been deleted from the data base. This same fact is unaffected by deleting Whiting's Employee tuple in Pers_Info_B. This irregularity in Pers_Info_A is called a deletion anomaly and again suggests advantages to be gained by partitioning a large relation into smaller relations. The fact that Whiting was the component manager of Finance means that an additional update is required in the Pers_Info_B Comp_mgr relation when a new manager is named.

There is one observation of Pers_Info_A and Pers_Info_B that suggests a disadvantage to the multi-relation data base. Consider the following selection expressions that retrieve the employee number of Akins' manager from Pers_Info_A and from Pers_Info_B:

Pers Info A

```
call dsl_$retrieve (dbi_1,
   "-range (E Employee)
    -select E.mgr_num
    -where E.name = ""Akins""", arg_1, code);
```

Pers Info B

```
call dsl_$retrieve (dbi_2,
   "-range (E Employee) (C Comp_mgr)
    -select C.mgr_num
    -where ((E.name = ""Akins"") &
      (E.comp = C.comp))", arg_1, code);
```

Not only is the selection expression simpler for the retrieval from Pers_Info_A, but the expected time to retrieve the manager's employee number is less for Pers_Info_A since the data base manager must search both relations in the Pers_Info_B data base in order to satisfy the -where clause. Thus, the data base administrator must carefully weigh a number of consequences when designing a data base that is optimal for a particular set of data base requirements. It is generally agreed, however, that the advantages of a partitioned data base outweigh the disadvantages.

Examples of Normalization

The process of normalizing a data base consists of a subjective design process (performed by the DBA) where complex relations are transformed into simpler relations without loss of information. Normalization has been formalized to the extent that there are three well-defined normal forms: first normal form (FNF), second normal form (SNF), and third normal form (TNF).

FIRST NORMAL FORM

The conversion of some collection of data into FNF is essentially the process of eliminating repeating groups and hierarchical structures: every attribute must be defined over a domain that is a relation containing no more than one attribute. For example, consider a collection of supplier data where each supplier references several projects. Such a collection can be represented as follows, where "supplier" and "project" can be thought of as arrays and the items enclosed in parentheses represent column headings:

```
supplier (supp_no name address zip project (proj_no qty_s mgr))
```

The project attribute is an array within an array. Such a construct may also be viewed as an hierarchical relationship, with the supplier being superior to the project. To convert such a structure to FNF, the attributes of project must be incorporated into the supplier relation:

```
supplier (supp_no* name address zip proj_no qty_s mgr)
```

where:

1. proj_no
   indicates the projects supported by a given supplier.

2. qty_s
   indicates the quantity of items supplied for a project by a given supplier.

3. mgr
   indicates the project manager.

Since a single supplier may supply more than one project, by definition proj_no is not functionally dependent on supp_no (see "NOTES" below).

SECOND NORMAL FORM

In order to ensure that each attribute is fully functionally dependent on its primary key (see "Notes" below), the supplier relation must be refined. For example, the quantity of items supplied (qty_s) is dependent upon both components of the primary key combination of supp_no and proj_no. However, the project manager attribute (mgr) is functionally dependent upon only one of the components: proj_no. In order to convert the data representation to SNF, a refinement of the supplier relation is required:

```
supplier (supp_no* name address zip)

supplier_proj (supp_no* proj_no* qty_s)

project (proj_no* mgr)
```

Thus, every nonprime attribute is fully functionally dependent upon the primary key to which it belongs.

NOTES: Functional dependence: an attribute (or group of attributes) B is functionally dependent upon A if each value of A never has more than one value of B associated with it. Alternatively, it can be said that, in such a case, A implies B.

Full functional dependence: B is fully functionally dependent upon a group of attributes A if B always depends upon all components of A and not upon any subset of A.

THIRD NORMAL FORM

Next, every attribute within the relation must be nontransitively dependent upon its primary key. In this connection, notice that, in the supplier relation, the address attribute is functionally dependent upon the supp_no attribute. That is, the supplier number (supp_no) implies the address. On the other hand, the supp_no attribute is not functionally dependent upon the address attribute. That is, the address of the supplier does not imply the supplier number. Furthermore, the zip attribute is functionally dependent upon the address attribute (i.e., address implies zip). This means that the zip attribute is transitively dependent upon the supp_no attribute (or the supplier number transitively implies the zip code of the supplier). To eliminate such transitive dependence, the following refinement of the supplier relation may be performed:

```
supplier (supp_no* name address)

address (address* zip)
```

NOTE: Transitive dependence: an attribute (or group of attributes) C is transitively dependent upon A if, at every instance, it is the case that,

C is functionally dependent upon B, and
B is functionally dependent upon A, but
A is not functionally dependent upon B.

An illustration of the above normalizing process applied against sample data values produces:

1. Unnormalized Data

```
supplier(supp_no name address zip   project(proj_no qty_s mgr))
         936     Acme Houston 44352         8       35    Jones
                                            3       10    Smith
                                            4       10    Smith
         909     Zula York    22369         8       12    Jones
                                            6       15    Gray
```

2. Third Normal Form

```
supplier (supp_no* name address)
          936      Acme  Houston
          909      Zula  York

project (proj_no* mgr)
         8         Jones
         3         Smith
         4         Smith
         6         Gray

supplier_project (supp_no* proj_no* qty_s)
                  936      8         35
                  936      3         10
                  936      4         10
                  909      8         12
                  909      6         15

address (address* zip)
         Houston  44352
         York     22369
```

Domains and Attributes

Reconsider the definitions:

1. Attribute: the name of a data field within a tuple.
2. Attribute value: the value of an attribute (data field) within a tuple.
3. Domain: the set of all values an attribute may assume.

Though this topic was not previously stressed, the data base administrator must consider the domain statement in the data model source as defining both a domain and a corresponding attribute having the same name as the domain. For the Pers_Info data base used earlier in this section, the emp_num domain is defined as a five-character string. The corresponding emp_num attribute would have a domain consisting of all integers from 00000 to 99999 if the domain had been declared fixed decimal (5) (not considering the sign).

Consider now the case when two or more attributes have the same domain as in the following data model source segment:

```
domain:    name         char(12),
           emp_num      char(5),
           comp         char(5),
           mgr_emp_num  char(5);

relation:  Employee (name emp_num* comp),
           Comp_mgr (comp* mgr_emp_num);
```

The data base corresponding to this data model is identical to the Pers_Info data base created in the previous tutorial with the clarifying exception that the employee numbers found in the Comp_mgr relation are now obviously the employee numbers of the managers of the components. Since the values of the emp_num and mgr_emp_num attribute are both taken from the same set of numbers, they are said to have the same domain. The following data model source establishes this relationship and is therefore a more proper definition.

```
domain:    name       char(12),
           emp_num    char(5),
           comp       char(5);

attribute: mgr_emp_num emp_num;

relation:  Employee (name  emp_num*  comp),
           Comp_mgr (comp*  mgr_emp_num);
```

The attribute statement defines a new attribute "mgr_emp_num" and equates its domain of values to that of the emp_num attribute. The two attributes are said to be "domain compatible," a condition required for successful use of the -inter, -differ, and -union set operators.

It is recommended that generic names be used for domains, such as char_5 for char(5). Then, the attribute names that are to be used in the relation statement can be defined via the attribute statement as was done in Step 1 of the MRDS tutorial for the Pers_Info.db data base.

Further, it is recommended that attribute names be unique across the entire data base, not just within each relation, so that any set of attributes can be selected and have unique names for temporary relation definition.

Primary and Secondary Indexes

Practical considerations force a data base administrator to be concerned with the storage requirements of the data base and the computer resources required when updating the data base. In order to optimize these considerations, a data base administrator requires some insight into the implementation of MRDS on Multics. The relation of a data base is implemented as an indexed sequential file, implying as the name suggests, that the accessing of a particular record (tuple) within the file proceeds either as a sequential search, record by record, or directly if an index of the record desired is provided. In general, each record of a file may have more than one index.

Within MRDS, the primary key of a relation becomes the primary index of the file. In other words, the asterisk (or set of asterisks) appearing in the data model relation statement designates the primary index of the corresponding file. (A set of asterisks specifies that a primary key is to be formed by concatenating the attributes designated with an asterisk.) Additional, or secondary indexes, may also be designated in the data model by an index statement:

```
domain:    name      char(12),
           emp_num   char(5),
           comp      char(5);

attribute: mgr_emp_num  emp_num;

relation:  Employee (name* emp_num comp*),
           Comp_mgr (comp* mgr_emp_num);

index:     Employee (comp  emp_num),
           Comp_mgr (mgr_emp_num);
```

In this example, the Employee file has comp and emp_num as secondary indexes and a primary index formed from the concatenation (joining) of the attributes name and comp. Concatenation of attributes form a larger primary key than would otherwise be formed and is done only to gain uniqueness in the primary key. In this case, the name attribute alone does not ensure uniqueness (e.g., two Smiths may work for the company; the data base designer in this case has determined that two Smiths do not work in the same component). The Comp_mgr file has the "mgr_emp_num" as the secondary index and "comp" as the primary index.

The following guidelines are suggested when designing a data base and deciding the number and type of indexes.

1. All relations must have one and only one primary key. The key may, however, be composed of several attributes joined together.

2. The primary key values must be unique. (For example, two employees working in the same component and having the same last name would result in a duplicate key error when the second employee's tuple is stored into the Employee relation.)

3. Secondary indexes are used to increase the efficiency of data base accesses. Secondary indexes are optional and should be used with discretion because of the increase in data base storage and update overhead.

4. Secondary indexes may only consist of individual attributes; they cannot be concatenated. An attribute selected as a secondary index need not have unique values. However, storage usage and update time increases with the higher number of duplicate values.

5. Selection performance is a function of the type of attributes used in the search of a relation. Several attribute types are grouped into three classes of decreasing performance:

   a. Attribute is the entire primary key

   b. Attribute is the most significant (leftmost) part of the primary key (called a key head) or is a secondary index

   c. Attribute is not the most significant part of the primary key and is not a secondary index

NOTE: The primary key in MRDS can be a maximum of 2277 bits long. Key attributes have a storage length as defined by their data type (i.e., fixed bin(17) aligned takes 36 bits). The total length of a key is determined by the sum of the lengths of the attributes making up the key.

The maximum length (mentioned above) also applies to any single attribute which is to be a secondary index.

A successful create_mrds_db with the -list option (or display_mrds_dm with the -long option) gives information on data bit lengths.

SECONDARY INDEXING

This example is based upon a situation that arises sometime after the creation of a hypothetical data base called AB_Company. The data model source of this data base is:

```
domain:   l_name      char(12),
          emp_num     char(5),
          location    char(5),
          component   char(5),
          salary      char(7),
          mgr_emp_num char(5);

relation: Employee (l_name emp_num* location component* salary),
          Comp_mgr (component* mgr_emp_num);
```

Assume that a new requirement demands frequent searches for the last name and employee number of all employees who have a salary of "x." Because salary is not a key of the Employee relation, the procedure to do this task would require a sequential search of the entire Employee relation to select those employees having the specified salary.

Placing a secondary indexing on the salary attribute eliminates the need to sequentially search the entire data base. The retrieve request simply proceeds as a direct (keyed) access thereby eliminating the sequential search. The data model source would require the additional statement:

```
index:  Employee(salary);
```

Situations of this nature should be discussed with the data base administrator in order to determine whether or not the data base should be redesigned to include a secondary index on the frequently searched attribute.

SECTION 3

COMMANDS

This section contains descriptions of the MRDS commands, presented in alphabetical order. Each description contains the name of the command, discusses the purpose of the command, and shows the correct usage. Notes and examples are included where necessary for clarity.

The following is a summary of MRDS commands.

adjust_mrds_db, amdb
: administrative tool for managing a data base's concurrent access control segment.

copy_mrds_data, cpmd
: copies data from one MRDS data base to another.

create_mrds_db, cmdb
: creates an unpopulated MRDS data base.

create_mrds_dm_include, cmdmi
: builds an include file of structure declarations suitable for use in accessing the data base from PL/I programs.

create_mrds_dm_table, cmdmt
: provides a picture or graphic display of the data model/submodel structure.

create_mrds_dsm, cmdsm
: creates a data submodel definition (provides an alternate view of the data base).

display_mrds_db_access, dmdba
: displays the effective security access to relation and attribute data provided by a given view of the data base.

display_mrds_db_population, dmdbp
: displays the current number of tuples stored in the relations of a given view of the data base.

display_mrds_db_status, dmdbs
: displays the open and concurrent users in the given view of a data base.

display_mrds_db_version, dmdv
: displays the version of a MRDS data model/submodel.

display_mrds_dm, dmdm
: displays specified information from the data model.

display_mrds_dsm, dmdsm
: displays specified information from the data submodel and optionally displays related data model information.

display_mrds_open_dbs, dmod
: displays a list of pathnames, opening indexes, and opening modes of all currently opened data bases in the user's process.

display_mrds_scope_settings, dmss
displays opening information and scope set for those openings for all data bases open in the user's process.

display_mrds_temp_dir, dmtd
displays the directory under which temporary storage for a given data base opening is placed.

mrds_call, mrc
provides a command-level interface to the MRDS Data Sublanguage (DSL) for data base development. For a complete description, see Section 9, "Data Base Development Tools."

quiesce_mrds_db, qmdb
an administrative tool that places the data base in a quiescent (non-active) state for such purposes as dumping, etc.

secure_mrds_db, smdb
provides the ability to turn on (or off) attribute level security features.

set_mrds_temp_dir, smtd
changes the current pathname of the directory that is used for temporary storage in the next call to dsl_$open.

unpopulate_mrds_db, umdb
a data base application development tool that deletes all data from a data base.

In examples that illustrate the user's interaction with the terminal, the lines typed by the user are indicated with an exclamation mark (!) to the left of the line to distinguish user entries from system output. This is for illustrative purposes only; the user does not actually type the exclamation mark. Input commands are expected to be on one line. This is accomplished (for lines longer than can be accommodated on the terminal) by utilizing the automatic wrap-around feature of most terminals. Comments that serve an explanatory purpose are included within a program by enclosing them within "/* comment */". Likewise, the examples do not show the escape carriage return ("\CR") and line feed ("\LF") required if the user were to actually input the commands on multiple lines as shown.

adjust_mrds_db

Name: adjust_mrds_db, amdb

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

Usage

amdb path {-control_args}

where:

1. path
   is the relative or absolute pathname of the data base whose concurrency control segment is to be manipulated. The .db suffix need not be given for new version data bases. This cannot be a submodel pathname.

2. control_args
   may be chosen from the following:

   -dead_procs, -dpr
   the data base control segment deletes information pertaining to dead processes (i.e., data base openers whose processes terminated without closing the data base). Non-passive dead processes (processes with some form of update scope set) may leave the data base in an inconsistent state.

   -force, -fc
   suppresses the query given for the -reset control argument.

   -no_force, -nfc
   allows the query for the -reset control argument to be given. (Default)

   -reset, -rs
   the data base control segment is re-established in a consistent state. If there are active users of the data base, the command queries the user whether to continue, since other active users lose concurrency control protection if this invocation proceeds. (Default)

   -trouble_switch state, -tsw state
   where state may be either "on" or "off". This sets the data base concurrency control trouble switch ON or OFF. If the switch is on, attempts to open the data base fail. This can be used to lock out users when there is a question about the data base integrity. The DBA can then restore damaged segments or rollback the data base to a consistent state.

Notes

The user must be a DBA to use this command.

The -reset and -dead_proc options may not be used together. The -force and -no_force control arguments, given without -reset, imply -reset.

The -reset option (default) should be used only after display_mrds_db_status is invoked, to determine if there are open users and to notify those users to close their opening of the data base. If open users are active during use of this option, they lose 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 cannot be used with the current version of MRDS, and adjust_mrds_db with the -reset option must be used on the data base in order to convert it to version 5 concurrency control. The current version of concurrency control may be displayed via display_mrds_db_status using the -long option.

Current users of r-s-m-d scope mode encodings do not have to change their application programs to use version 5 concurrency control. Application programs calling dsl_$set_scope or dsl_$set_scope_all which use the old r-u scope mode encodings need be changed to the encodings described in this manual (e.g., 2 no longer means s-m-d, just s).

Examples

```
! display_mrds_db_status foo -long

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

! mrds_call open dmdm update

    Error:  mu_concurrency_control error by >unb>bound_mrds_!2232.  The data
    base 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>foo.db -reset" must be run before it
    can be used.

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

! adjust_mrds_db foo -reset
! mrds_call open foo update

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

adjust_mrds_db

```
! display_mrds_db_status foo -long

   Concurrency control version:  5
                Data base path:  >udd>m>jg>dr>foo.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 foo

   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>foo.db data base??

! no

! display_mrds_db_status foo

                Data base path:  >udd>m>jg>dr>foo.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 foo -dead_procs

! display_mrds_db_status foo

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

Name: copy_mrds_data, cpmd

This command copies data from one MRDS data base to another.

Usage

cpmd input_db_path output_db_path {-control_args}

where:

1. input_db_path
   is the pathname of the data base from which data is copied. If the pathname does not have a suffix of db, then one is assumed. However, the db suffix must be the last component of the name of the input segment.

2. output_db_path
   is the pathname of the data base to which date is copied. The data base must already exist. If the pathname does not have a suffix of db, then one is assumed. However, the db suffix must be the last component of the name of the output segment.

3. control_args
   can be chosen from the following:

   -input_prevent_ops OPS
   specifies the prevent scope on the input relation(s), where OPS is the set of operations that the user wishes to deny other openers of the input data base for the relation(s) being copied. (Default is "dms" --refer to Notes for a list of scope mode abbreviations.)

   -output_prevent_ops OPS
   specifies the prevent scope on the output relation(s), where OPS is the set of operations that the user wishes to deny other openers of the output data base for the relation(s) being copied. (Default is "dms" --refer to Notes for a list of scope mode abbreviations.)

   -relation RELNAME, -rel RELNAME
   specifies that RELNAME be copied. Only one relation at a time can be copied using this control argument. If this control argument appears more than once in a command line, the previous occurrence is overridden.

   -transaction_group_size N
   specifies copying N tuples within the confines of a single transaction. If this control argument is omitted, or if N is equal to 0, then each access to a protected data management file is completed as a separate transaction.

Notes

The abbreviations used for prevent scope operations (for either input or output) are as follows:

a append_tuple

s append_tuple (same as a)

copy_mrds_data

d delete_tuple

m modify_attr

n null

r read_attr

u update (same as dms)

The prevent scope is made up of a concatenation of the desired operation abbreviations. If "n" prevent scope is given, then no other mode may be specified for that prevent. Each of the other modes may be used only once in the same prevent scope.

Relations that are copied must be identical in their makeup, having the same attributes, attribute names, indexes, etc. It is suggested, where possible, that both data bases be created using the same create_mrds_db source. When using the -relation control argument however, it is possible to copy from data bases with differing models, as long as the relation being copied is the same in both data bases.

Name: create_mrds_db, cmdb

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

Usage

cmdb 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. (See Data Model Source below.)

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 the db suffix is added automatically if not given with the argument. See Architecture of the Data Base below.)

3. control_args
   may be chosen from the following:

   -data_management_file {STR}, -dmf {STR}
   creates relation data files that are manipulated by the Multics Data Management System. STR is an optional mode string that defines the characteristics of the data management files. This mode string applies to all relations created in the data base. See Notes for a list of valid modes.

   Access required: The directories under which the listing segment and the data base directory are to be created must have append access for the user, similarly for the temp_dir if used. The containing directory access must be "sm", if -force is used.

   -force, -fc
   causes an existing data base of the same pathname as the given or default pathname to be deleted and this new data base to be created in its place.

   -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 list (rather than cmdb) as the suffix.

   -no_force, -nfc
   does not allow a data base of the same pathname as the given or default pathname to be created when such a data base already exists. (Default)

-no_list, -nls
indicates that no listing is to be created. (Default)

-no_secure
causes the data base to be created in the unsecured state. (Default)

-secure
causes the data base to be created in the secured state. See the secure_mrds_db command for details on the secured state. Also refer to Section 7 for information on the effect of the secured state on commands and subroutines.

-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 256 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.

-vfile, -vf
creates relation data files that are manipulated by vfile_. (Default)

Notes

The largest data base that can be created is 256 relations. MRDS allows 256 attributes per relation.

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

The person who invokes the create_mrds_db command automatically becomes a DBA for the data base created since the creator of a data base is always given "sma" access to the data base directory. The invoker of create_mrds_db needs "a" access to the directory that contains the data base. If -force is used to remove an existing data base, "sm" access is also required.

List of modes (for use with -data_management_file control argument):

protection
creates relations as protected data management files. Relations created with this mode can be accessed only if the process is in a transaction.

concurrency
provides concurrency control when accessing relations. This mode is valid only if protection is enabled.

rollback
provides rollback before images are taken when updating a relation. This mode is valid only if protection is enabled.

create_mrds_db

If the mode appears in the mode string preceeded by "^", then the mode is set to off. In the case of duplicate mode specifications, the last mode specified takes effect.

The default for protection is on. If protection is on, the default for concurrency and rollback is also on. If protection is off, the default for concurrency and rollback is off. If no mode string is specified in the -data_management_file control argument, a default mode string of "protection,concurrency,rollback" is used.

create_mrds_db

Data Model Source

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_name_1 (
               i_rel_1_i_attr_1 ... i_rel_1_i_attr_L),
                         .
                         .
          indexed_relation_name_N (
               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, with only spaces separating attribute names within a relation.

Statement Usage

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 attribute and index statements are optional.

The domain statement defines the data type that any attribute defined over that domain is to have. Any legal PL/I 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 Programmer's Reference Manual for a description of Multics data types. When data needs to be converted from the user's type into the storage type declared in the domain statement, the subroutine assign_ is used. See Subroutines and I/O Modules, Order No. AG93 for a description of data types supported by that routine.

The relation statement takes previously defined attributes and defines the relations that are to exist in the data base. 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 "Data Base Design" in Section 2.) There may be up to a total of 256 different key and non-key attributes in any one relation. Up to 256 different relations may be defined.

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.

The index statement 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, "Data Base Design".) The same key length

create_mrds_db

restrictions apply to each single inverted attribute as apply to the total primary key. The first attribute of a multi-attribute primary key may be used as if it were a secondary index.

Formatting Data Model Source

The keywords domain, attribute, and relation may be 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 by giving the source segment an add_name with .pl1 suffix and using indent, or by creating the data base first and 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.

-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 user's 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.

-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.

-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.

See Appendix E "Administrator Written Procedures" for a detailed explanation of the interface and examples of how these options may be used.

DATA BASE ARCHITECTURE

The data base is a directory with the identifying suffix ".db". This directory contains the following:

NAME	TYPE	PURPOSE
resultant_segs_dir	directory	place for copy of mrds internal structure for speedup of open
db.control	segment	concurrency control
db_model	segment	domain information, relation names
{relation_name}.m	segment	model of the relation structure
{relation_name}	file	relation data storage
secure.submodels	directory	place for secure submodels

Note: There are two segments (dbcb and rdbi) under the resultant_segs.dir directory. The dbcb and rdbi segments are the copies of internal structures.

There is one relation model segment and one relation data file for each relation defined in the data base.

Examples

```
! print x.cmdb

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

dom: a bit;  /* simplest possible data base */
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>d>dbmt>ndb>mike>x>doc>x.cmdb
          Created by:         Kubicar.Multics.a
          Created on:         01/16/84  1412.7 mst Mon
          Data base path:     >udd>d>dbmt>ndb>mike>x>doc>x.db
                 Options:     list


     1          dom: a bit;  /* simplest possible data base */
     2          rel: b(a*);

NO ERRORS

DATA MODEL FOR VFILE DATA BASE  >udd>d>dbmt>ndb>mike>x>doc>x.db
```

create_mrds_db

```
Version:                4
Created by:          Kubicar.Multics.a
Created on:          01/16/84  1412.8 mst Mon

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


RELATION NAME:    b
Number attributes:          1

   ATTRIBUTE:

      Name:       a
      Type:       Key
      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), /* longitude + latitude */

          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
```

create_mrds_db

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)
```

Name: create_mrds_dm_include, cmdmi

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

Usage

cmdmi path {-control_args}

where:

1. path
   is the relative or absolute pathname of the data base model or submodel, with or without suffix. It requires "r" ACL to the data model. If the data base is secured, then the path must refer to a submodel in the secure.submodels directory under the data base, unless the user is a DBA. If a suffix is not supplied and both a model and submodel exist in the same directory, then the model is found before the submodel.

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

   -based
   specifies that the resulting include file structure declaration has the "based" PL/I attribute.

   -no_based
   specifies that the resulting include file structure declaration does not have the based attribute. (Default)

   -order rel_name1 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.

   -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.)

create_mrds_dm_include

Notes

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

<entryname of the input path with the db or dsm suffix removed>.incl.pl1

If the segment does not exist, it is created.

If the data base is secured and the user is not a DBA, then the "key" comment on attributes is changed to "indexed" for the key head attribute and remaining key attributes have no comments.

If a -decode_declare option exists on an attribute domain, then the declaration appears in the include file since this is the user view and the data base storage data type is not of use.

Examples

```
! display_mrds_dm foo -cmdb

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

domain:
          data
               real float decimal (10) aligned /* 9-bit */
               -decode_dcl character (20) varying aligned,
               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

```
! create_mrds_dm_include foo -based
! pr foo.incl.pl1

/* ********************************************
   *                                          *
   * BEGIN foo.incl.pl1                       *
   *      created: 02/24/81  1407.2 mst Tue   *
   *           by: create_mrds_dm_include (3.0) *
   *                                          *
   * Data model >udd>m>jg>dr>foo.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 foo.incl.pl1    ******************************/


! display_mrds_dm foo -cmdb


 /* Created from   >udd>m>jg>dr>foo.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

```
!  create_mrds_dm_include foo -order rel_2 rel_1
!  print foo.incl.pl1

   /* ********************************************
      *                                          *
      * BEGIN foo.incl.pl1                       *
      *       created: 03/16/81  1321.1 mst Mon  *
      *            by: create_mrds_dm_include (3.0) *
      *                                          *
      * Data model >udd>m>jg>dr>foo.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 foo.incl.pl1    *********************************/
```

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 base 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

cmdmt path {-control_args}

where:

1. path
   is the relative or absolute pathname of the data model/submodel of the data base, with or without the suffix. The user must have "r" access to some relation in the data base. The pathname must be the first argument. If the data base is secured, then the path must refer to a submodel in the secure.submodels directory under the data base, unless the user is a DBA.

2. control_args
   can 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.

   -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)

   -long, -lg
   causes the PL/I data type information to be displayed below each attribute name, inside each box. (Default)

   -order rel_name1 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 is 59 lines)

create_mrds_dm_table

Notes

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

<entryname of the input path with the db or dsm suffix removed>.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 data base is secured and the user is not a DBA, then the key head attribute is marked as "indexed" and remaining key attributes are unmarked.

If a -decode_declare option exists on an attribute domain, then the declaration appears in the table since this is the user view and the data base storage data type is not of use.

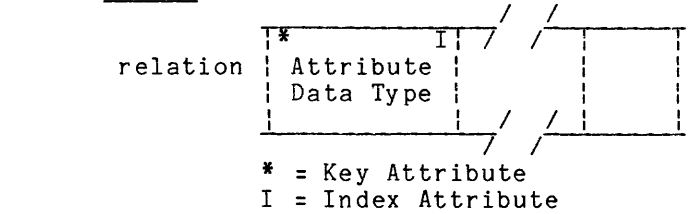
Examples

```
! display_mrds_dm cmdmt -cmdb

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

domain:
          data
               real float decimal (10) aligned /* 9-bit */
               -decode_dcl character (29) 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 foo -line_length 65
! print foo.table
```

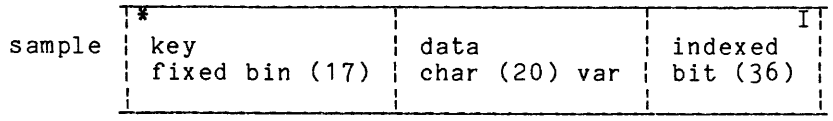
create_mrds_dm_table

```
/* ****************************************
   *                                      *
   * BEGIN foo.table                      *
   *       created: 02/26/81  1158.6 mst Thu *
   *            by: create_mrds_dm_table (3.0) *
   *                                      *
   * Data model >udd>m>jg>dr>foo.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 foo -brief -order rel_2 rel_1
!  print foo.table

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

    LEGEND:
                 _____________/ /________
                 |*         I|/ / |      |
       relation  | Attribute |    |      |
                 |___________/ /__|______|
                            / /
                  * = Key Attribute
                  I = Index Attribute

             ________
             |*      |
   rel_2     | number|
             |_______|

             ______
             |*    |
   rel_1     | char|
             |_____|
```

Name: create_mrds_dsm, cmdsm

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

Usage

cmdsm 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. (See "Data Submodel Source" below.)

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
   can be chosen from the following:

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

   -install, -ins
   creates the submodel in the secure.submodels directory that is under the data base directory rather than in the working_dir (see "Data Base Architecture" under the create_mrds_db command). The use of this control argument causes a directory named secure.submodels to be created under the data base directory if it does not already exist. This control argument is restricted to DBAs (see Section 7).

   -list, -ls
   creates a segment containing a listing of the submodel source, followed by information about the submodel to model mapping, in the working directory. The segment also contains a list of any errors found while creating the submodel.

   -no_force, -nfc
   if a submodel with the same name already exists, queries the user as to whether it can be overwritten. This control argument undoes the effects of a -force. (Default)

   -no_install, -nins
   creates the submodel in the working_dir. (Default)

   -no_list, -nls
   specifies that a listing segment is not created. (Default)

create_mrds_dsm

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 data base. If the data base 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 twofold: to map the user's view of the data base into the actual data base description (i.e., the data model) and 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);
```

Take 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 in which the statements must occur.

create_mrds_dsm

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 data base view. Multiple relation statements can occur provided each model relation is used to define, at most, one submodel relation.

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 view of a 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 30 characters long and 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 32 characters long and may be composed of letters, numbers, hyphens, and underscores, but must begin with a letter.

create_mrds_dsm

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 data base associated with the submodel is a secure data base. (See the secure_mrds_db command and Section 7 "Security".)

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 data base model can open the associated data base and is subject to the access privileges specified in that submodel. A person can 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 for which they define access. Only one default relation access and one default attribute access statement may 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, which 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;
```

create_mrds_dsm

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, which 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 clause" (attribute access control list) can be considered a default attribute access statement which is in effect only over the associated 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 nor deleted from the relation.

Note that any form of the access keywords may be used in the access control list. A null access cannot be specified with any other access. The order of a combination of append_tuple and delete_tuple is not important. Currently there is the restriction that append_tuple and delete_tuple may 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).

create_mrds_dsm

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 may not be read or modified.

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

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

Examples

The following examples show different submodels which are all defined over the States data base described in the examples of the create_mrds_db command. The first submodel is a full view 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 data base
     with no name changes.  Since no access is specified, the default relation
     access of null and 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 state_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 relation */
          person
             (ln = last_name  /* last name of person */
              fn = first_name /* first name of person */
              salary          /* person's salary */
              expenses        /* expenses of person to date */),


     /* location of person */
          person_state        /* state of residence */
             (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.  Notice as well 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:
                 last_name (r),
                 first_name (r);

     rel_acc:
                 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. Invoking the command using no control arguments 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
```

The following invocation creates the submodel in the secure.submodels directory under the states.db directory. Only a DBA can use this control argument.

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

The following invocation installs the submodel in the secure.submodels directory and writes over any existing cmdsm_source_example_3 without querying the invoker.

```
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 overwriting 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 -ins
```

Name: display_mrds_db_access, dmdba

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

Usage

```
display_mrds_db_access path {-control_args}
```

where:

1. path
   is the relative or absolute pathname of a data base model or submodel, with or without suffix, that supplies the view for which the user wishes to see access information. If both a data model and submodel of the same name are in the same directory, then the model will be found if no suffix is given.

2. control_args
   may be one of the following:

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

   -long, -lg
   specifies that all information related to access be displayed. (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. This control_arg must appear after path.

Notes

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

Control arguments can be overridden, in that the last one specified takes effect (e.g., -bf followed by -lg implies -lg).

This command only works for version 4 data bases.

The Multics system ACLs, the MRDS access modes, and the result of these two, an effective access, is displayed for each relation and attribute in the given view. Access modes displayed depend on the secured state of the data base as follows:

display_mrds_db_access

DB SECURED STATE	MODES
off	r-e-w
on	r-a-m-d

The r-e-w refers to Multics ACLs. The r-a-m-d refers to the new attribute level security related operations of read_attr, append_tuple, modify_attr, and delete_tuple respectively.

Examples

```
! display_mrds_db_access submodel

Data base path: >udd>m>jg>dr>model.db
      version:  4
                data base is in a secure state.

Submodel path:  >udd>m>jg>dr>model.db>secure.submodels>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         rw       n          n
              d001         rw       r          r


! display_mrds_db_access submodel -brief

r001                       n
              k001         r
              d001         n

r002                       d
              k001         m
              x001         n
              d001         r
```

display_mrds_db_population

Name: display_mrds_db_population, dmdbp

This command displays the current tuple count for each relation in the given data base 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_args}

where:

1. path
   is the relative or absolute pathname of the data base model or submodel, with or without suffix, that is to have the relation's population statistics displayed. If both a data model and submodel of the same name are in the same directory, then the model will be found if no suffix is given.

2. control_args
   may be one of the following:

   -brief, -bf
   limits the output display to only relation names and their current tuple count. (Default)

   -long, -lg
   displays the average number of tuples selected by index during retrieval.

   -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. This control_arg must appear after path.

Notes

Version 3 data bases must have been opened at least once for exclusive update. They cannot have secondary index information displayed.

For version 4 data bases, only a DBA may use this command on a secured data base, with 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".

*

display_mrds_db_population

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

  Displaying version 4 data model: >udd>d>dbmt>ndb>K>x>doc>x001.db

  RELATION  TUPLES    INDEX     AVE TUPLES SELECTED

  r001      100
                      x001      16
```

A description of the -long form output follows:

Relation: the name of the relation in the user's view.

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

Index: the indexed attributes in the relation.

Ave Tuples Selected: the number of tuples that MRDS expects to retrieve when doing a comparison on an indexed attribute. This estimate is based on the number of duplicate keys for that index.

This page intentionally left blank.

display_mrds_db_status

Name: display_mrds_db_status, dmdbs

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

Usage

display_mrds_db_status path {-control_args}

where:

1. path
   is the relative or absolute pathname of the data base, or of a submodel defined for that data base, for which concurrency control information is desired. If both a data model and submodel of the same name are in the same directory, the model will be found if no suffix is given.

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 data base.

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

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

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

Notes

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 "Activation: normal". Only exception conditions or necessary information are displayed (e.g.,

display_mrds_db_status

"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

```
! 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 data base "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
```

display_mrds_db_version

Name: display_mrds_db_version, dmdv

This command displays the MRDS data model/submodel version, creator, and creation time.

Usage

dmdv path

where path is the pathname of the data model/submodel version to be displayed (with or without the db or dsm suffix).

Notes

This command requires access to open the data model or submodel for retrieval (for example, as in mmi_$open_model or msmi_$open_submodel). (See dsl_$get_path_info for a subroutine interface.)

If a data base model and submodel of the same name are in the same directory, the model is found if a suffix is not given.

Example

```
! display_mrds_db_version CS_III

   Data model: >udd>Demo>demt>db7>jg>CS_III.db
      version: 4
      created: 02/01/80 1419.0 mst Fri
           by: JGray.Multics.a
```

display_mrds_dm

Name: display_mrds_dm, dmdm

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

Usage

dmdm 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
   can be chosen from the following:

   -attribute {modifier}, -attr {modifier}
   displays attribute information. The modifier may be name(s) or -unreferenced (-unref). If name(s) is supplied, information for the attribute name(s) is displayed. If -unreferenced is supplied, attribute information anbout all unreferenced attributes is displayed. If no modifier is supplied, attribute information about all attributes is displayed.

   -brief, -bf
   displays only relation and attribute names. No information on the characteristics of the attributes and relations is provided. This control argument is incompatible with the -names control argument.

   -cmdb
   specifies that the output is to be in the same format as an input source text for create_mrds_db. If the -output_file control argument is supplied, then the segment can be used to create another data base with the same definitions. Only the -brief, -long, and -output_file control arguments are compatible with this control argument.

   -crossref {type}, -xref {type} displays an information cross-reference. The type may be domain (dom), attribute (attr), or all. If the type is domain, each domain is listed with a list of attributes in which the domain is referenced. If the type is attribute, each attribute is listed with a list of relations in which the attribute is referenced. If the type is all, both domain and attribute cross-references are displayed. (Default is "all".) See the examples below which show the information displayed.

   -domain {modifier}, -dom {modifier}
   displays domain information. The modifier may be name(s) or -unreferenced (-unref). If name(s) is supplied, information for the domain name(s) is displayed. If -unreferenced is supplied, domain information about all unreferenced domains is displayed. If no modifier is supplied, domain information about all domains is displayed.

   -header, -he
   displays header information for the data base.

-history, -hist
displays restructuring history information. If the data base is restructured more than once, the history entries are displayed in reverse chronological order.

-index names, -ix names
displays information about indexed relations for each relation name supplied. If no names are supplied, then information about all indexed relations is displayed.

-long, -lg
displays all available information about relations and their attributes. For relations, this includes the number of attributes and the layout of the attributes in the tuple. For attributes, this includes the name of the underlying domain and the declaration. This control argument is incompatible with the -names control argument.

-names, -nm
displays the format of domains, attributes, relations, and indexed relations as a list of the names. This argument is incompatible with -brief or -long control arguments.

-no_header, -nhe
prevents display of the header information. (Default)

-no_output_file, -nof
displays output on the user_output switch. (Default)

-output_file path, -of path
places the output in the segment named by path rather than being displayed on the user_output switch. If the segment already exists, its contents are overwritten.

-relation names, -rel names
displays relation information for each relation name supplied. If no names are supplied, the relation information for all relations is 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 source may require 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.

Notes

If neither -long nor -brief is specified, the relation name is displayed for each relation as well as the name and user view declaration of each attribute.

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

display_mrds_dm

For version 4 data bases, the user must be a DBA in order to use this command on a secured data base.

If -long is specified, the header output indicates the secured state of the data base.

Examples

```
! display_mrds_dm dmdm.db -long

   DATA MODEL FOR VFILE DATA BASE  >udd>d>dbmt>ndb>K>x>doc>dmdm.db
   Data base secured.

   Version:                4
   Created by:          Kubicar.Multics.a
   Created on:          01/16/84  1515.6 mst Mon

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


   RELATION NAME:    sample
   Number attributes:          3

      ATTRIBUTES:

         Name:       key
         Type:       Key
         Domain_info:
               name: key
               dcl:  character (1) nonvarying aligned


         Name:       data
         Type:       Data
         Domain_info:
               name: data
               dcl:  character (1) nonvarying unaligned


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

display_mrds_dm

```
! 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 -names

            sample
```

display_mrds_dsm

Name: display_mrds_dsm, dmdsm

This command displays information about the specified MRDS data submodel.

Usage

dmdsm 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 data submodel file name.

2. control_args
   can be chosen from the following:

   -access, -acc
   specifies that access information (both relation and attribute) is to be displayed.

   -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 data base. 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 data base 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 data base, 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.

   -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 the specified 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. (The default displays all relations.)

Example

The following examples all use the submodel example_4, which was generated from the example cmdsm_source_example_4 in the discussion of the create_mrds_dsm command. The submodel secure_example_4 is the same submodel defined for a secure data base.

```
! display_mrds_dsm example_4

   a     person
   r       last_name
   r       first_name
   n       salary
   r       expenses

   d     person_state
   r       last_name
   r       first_name
   rm      key

   ad    state_history
   r       key
   rm      state_name
   rm      date_time
   rm      text


! 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

 Data base path:      >udd>Multics>examples>states.db
       Version:       4
```

display_mrds_dsm

```
    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 delete_tuple

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

display_mrds_dsm

```
          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, delete_tuple);

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

display_mrds_dsm

```
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

   a    person
   d    person_state
   a    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

 Data base 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 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

```
! 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

 Data base 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)
```

display_mrds_open_dbs

Name: display_mrds_open_dbs, dmod

This command displays the data base opening indexes, opening modes, and pathnames of all model and submodel openings of data bases currently open in the user's process.

Usage

```
display_mrds_open_dbs
```

Note

The output is a formatted list of openings, in opening index order, which contains the opening model or submodel path and the mode in which the opening was obtained. Data base and submodel suffixes are shown whether or not they were used in the call to open. 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:
            >udd>m>jg>dr>model.db
            update
            >udd>m>jg>dr>submodel.dsm
            retrieval


 ! display_mrds_open_dbs

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

display_mrds_scope_settings

Name: display_mrds_scope_settings, dmss

This command displays concurrency control scope mode information for all currently open data bases in the user's process. The versions of the concurrency control, data base, and submodel (if used for the opening) are displayed, as well as the absolute paths of the data base and the submodel (if used for the opening). The opening mode is also displayed.

Usage

display_mrds_scope_settings

Note

All versions of data base scope settings may be displayed with r-s-m-d modes used for version 3 and earlier data bases, and read_attr, modify_attr, append_tuple, and delete_tuple (abbreviated as r-m-a-d) used for version 4 data bases 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.)

Example

```
! 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
                    data base model path:  >udd>m>jg>dr>mod_del.db
                       data base version:  4

            Relation                              Permits   Prevents

            r001                                    ramd      ramd
            r002                                    ramd      ramd

   Opening index:  2
           mode:  exclusive_update

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

display_mrds_scope_settings

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

Relation                              Permits    Prevents

part                                    ramd       ramd
reorder                                 ramd       ramd
```

display_mrds_temp_dir

Name: display_mrds_temp_dir, dmtd

This command displays the directory under which temporary storage for a given data base opening is placed. This storage includes the "resultant model" that is created at open time for allowing access to the data base, storage for temporary relations, and intermediate results of complex searches. The default is the process directory.

Usage

dmtd temp_dir_indicator

where temp_dir_indicator must be one of the following:

1. database_index
   the opening index returned by the dsl_$open subroutine. If this option is used, then the temporary directory pathname for that particular opening is displayed.

2. -current, -cur
   displays the current temporary directory pathname that is used in subsequent calls to open.

Notes

To change from the default the command set_mrds_temp_dir is used to allow for the opening of a data base with a very large resultant model that does not fit in the process directory, for a data base with a large number of temporary relations, or for searches involving many tuples in several relations. This would be the case if a record quota overflow occurred in the process directory on a call to open.

See dsl_$get_temp_dir for a subroutine interface.

Example

```
! display_mrds_temp_dir -current

   >process_dir_dir>!BPNCndKBBBBBBB

! set_mrds_temp_dir >udd>m>cp>jg>1

   mrds_call open dept_store eu


   Open data base is:
   1         >udd>m>JGray>dr>dept_store
             exclusive_update

! display_mrds_temp_dir 1

   The temporary directory for data base index 1 is:

        >udd>m>cp>jg>1
```

quiesce_mrds_db

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

qmdb database_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:

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

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

   -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 zero seconds.

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. Only a DBA can use 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 data
  base path ">udd>m>jg>dr>qmdb.db".

! mrds_call  close  -all
```

quiesce_mrds_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
```

secure_mrds_db

Name: secure_mrds_db, smdb

This command provides the ability to turn on (or off) the attribute level security control features of MRDS. This is done on a data base basis. The secured state of a data base 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 data base to be secured, unsecured, or have its secured state displayed. The data base suffix need not be given. The path must be to a version 4 data base, not to a submodel.

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

   -display, -di
   causes the current data base secured state to be displayed without affecting that state.

   -reset, -rs
   causes the specified data base to be unsecured, regardless of its current secured state.

   -set
   causes the specified data base to be secured, regardless of its current secured state. (Default)

Notes

A data base that has been secured can be opened by a non-DBA, only via a submodel residing in the "secure.submodels" directory underneath the data base directory. This allows turning on (or off) attribute level security, which is implemented via submodel views, using their access control modes (version 5 submodels). Data bases earlier than version 4 are not supported.

This command requires the user to be a DBA. Once the data base has been secured, commands that normally operate against the model view requires the user to be a DBA. In addition, once the data base has been secured, commands using a submodel view require non-DBAs to use secured submodels.

See the documentation for create_mrds_db -secure, create_mrds_dsm -install, mmi_$get_secured_state, mmi_$get_authorization, and Section 7, "Security".

secure_mrds_db

Examples

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

set_mrds_temp_dir

Name: set_mrds_temp_dir, smtd

In the next call to dsl_$open this command changes the current pathname of the directory that is used for temporary storage. The temporary storage used is for the "resultant model" built during open time, for temporary relation storage, and for intermediate search results. The initial default for this directory is the process_dir. This command need only be used prior to the particular opening where a very large resultant model is built, large temporary relations are to be defined, or searches involving many tuples in several relations are to be done. A record quota overflow in the process directory during a call to dsl_$open, dsl_$retrieve, or dsl_$define_temp_rel indicates this need.

Usage

set_mrds_temp_dir directory_path

where directory_path is the relative or absolute pathname of a directory with more quota than the current temporary directory. The initial default is to use the process directory.

Notes

The temporary directory may be changed between calls to dsl_$open, thus resulting in different temporary directories for each opening. These may be displayed via display_mrds_temp_dir.

This command should only be used to avoid a record quota overflow in the process directory upon a call to dsl_$open, dsl_$retrieve, or dsl_$define_tem_rel. If a record quota overflow occurs in one of these calls, do a new_process, then set_mrds_temp_dir with a pathname of a directory that has more quota. If another record quota overflow occurs in that directory, set_mrds_temp_dir can be used again giving a directory with even more quota.

See dsl_$set_temp_dir for a subroutine interface.

unpopulate_mrds_db

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 data base that is to have all tuples in all relations deleted.

2. control_args
   may be one of the following:

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

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

Notes

Only a DBA can 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.

unpopulate_mrds_db

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 data base ">udd>m>jg>dr>test.db"?

! yes

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

   Data deletion complete, closing data base.

! display_mrds_db_population  test  -bf

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

      RELATION                                TUPLES

        r001                                      0
        r002                                      0
```

Name: update_mrds_db_version, umdbv

This command is used to convert existing populated (loaded) data bases to the most recent version of (new architecture) MRDS data bases as described in this manual. Since older MRDS data bases can be accessed with the new MRDS software, this command is not mandatory. However, to take advantage of improvements in MRDS, it is recommended that this command be used (no application program changes are necessary to use the updated version of the data base). A limited amount of restructuring is provided by this command. The end result is a populated new version data base (with the old data base unaffected).

Usage

update_mrds_db_version old_db_path new_db_path

where:

1. old_db_path
   refers to an existing old version populated data base (must be a data base whose version is 1, 2, or 3).

2. new_db_path
   refers to a new version unpopulated data base as created by the new version create_mrds_db command from a source identical to that of the old data base create_mrds_db source. This pathname must not be the same pathname as old_db_path.

Notes

The pathnames must include suffixes, if they exist, for the data base.

The header information for both old and new data bases is displayed, as well as the relation names and number of tuples moved for each relation.

Limited restructuring is possible, i.e., the secondary indexing of a data base may be altered using this command by defining the new data base index statement differently from the old data base index statement. Names of domains for attributes may also change via use of the attribute statement as long as the domain declarations remain the same.

The user can re-create the create_mrds_db source of the old data base by using the display_mrds_db command.

If an error occurs while populating the new data base, the new data base must be deleted and re-created with the create_mrds_db command after correcting the cause of the error. (update_mrds_db expects the new data base to be unpopulated when it is invoked.)

If the version 4 data base is secured, the user must be a DBA.

update_mrds_db_version

Example

```
! update_mrds_db_version >udd>XYZ>Doe>dept_store dept_store_reindexed.db
  UMDBV

  Opening data model:  >udd>XYZ>Doe>dept_store
       created: 06/12/79 1017.1 mst Tue
       version: 3
       by: Doe.SiteSA.a

  Opening data model: >udd>m>jg>dept_store_reindex.db
       created: 01/02/80  1215.2 mst Mon
       version: 4
       by:  JGray.Multics.a

  For relation "class", the number of tuples moved = 22
  For relation "emp", the number of tuples moved = 25
  For relation "loc", the number of tuples moved = 7
  For relation "sales", the number of tuples moved = 26
  For relation "supply", the number of tuples moved = 29

  Update complete, closing data models.
```

SECTION 4

DATA SUBLANGUAGE SUBROUTINES

This section describes those subroutine entries in MRDS which correspond to the functions described in the Data Sublanguage (DSL) in Section 2. These entries provide the user with the capabilities to:

- Open and close a data base
- Declare user-defined functions for use with the data base
- Retrieve data based on a flexible selection capability
- Modify and delete items within a data base
- Store new information into the data base
- Obtain information about the user's view of the data base
- Perform all of the above while allowing for concurrent access capability

FORMAL DEFINITION OF THE SELECTION EXPRESSION

Several of the DSL entries require a selection expression as an input parameter. Such an expression is a character string that precisely describes the data items in the user's view of the data base (the data model or the data submodel) to be manipulated. This character string may be a constant or a variable declared character varying or non-varying.

Formal Syntax

A formal syntax for MRDS is presented below using a metalanguage derived from Backus-Naur Form. The metalanguage symbols are defined as:

<> denotes a syntactical construct

::= means "is defined as"

[] denotes zero or one occurrence of (optional)

... denotes one or more occurrences of

* denotes key attribute

¦ denotes the logical inclusive "OR"

The inclusion of an underscore character under any of the symbols (see <bool_op> below) distinguishes that symbol as not being a part of the metalanguage but as being a part of the MRDS syntax. The character "^" preceding any symbol (see <qualifier> below) implies "not."

```
<selection_expression> ::=
       -another ¦ -compiled ¦ <select_set> ¦ <current_expression>

<current_expression> ::=
       -current <tuple_item> [<tuple_item> ...]

<select_set> ::=
       <alpha_expression> ¦ (<select_set>) <set_op> (<select_set>)

<set_op> ::=
       -union ¦ -inter ¦ -differ

<alpha_expression> ::=
       <range_expression> <tuple_expression> [<qualifier_expression>]

<range expression> ::=
       -range [<-no_ot ¦ -no_optimize>] [<-print_search_order> ¦ <-pso>]
       <range_definition> [<range_definition> ...]
       [<-print_search_order> ¦ <-pso>] <range_definition> [<range_definition> ...]

<range_definition> ::=
       (<tuple_variable> <relation>)

<relation> ::=
       <identifier> ¦ <temp_rel_index>

<temp_rel_index> ::=
       <argument_substitution>

<tuple_variable> ::=
       <identifier>

<identifier> ::=
       <letter> [<letter> ¦ <digit> ¦ _ ¦ - ]...

<tuple_expression> ::=
       -select <tuple_item> [<tuple_item> ...] ¦ <non_set_op_retrieve_expression>

<non_set_op_retrieve_expression> ::=
       -select -dup <tuple_item> [ <tuple_item> ...]

<tuple_item> ::=
       <tuple_variable> ¦ <tuple_attribute>

<tuple_attribute> ::=
       <tuple_variable>.<attribute> ¦ <temp_rel_key>

<temp_rel_key> ::=
       <tuple_variable>.<attribute>*

<attribute> ::=
       <identifier>

<qualifier_expression> ::=
       -where <qualifier>

<qualifier> ::=
       <term> ¦ ^(<qualifier>) ¦ (<qualifier>) <bool_op> (<qualifier>)

<term> ::=
       <expr_or_attr> <rel_op> <expr_or_attr> ¦ <expr_or_attr>
       <rel_op> <literal_constant> ¦ <expr_or_attr> <rel_op>
       <literal_argument_substitution>

<literal_argument_substitution> ::=
       <argument_substitution>

<expr_or_attr> ::=
       <tuple_attribute> ¦ [<expr>]
```

```
<expr> ::=
       <function> | <arith_expr>

<function> ::=
       <fn_name> (<arg_list>)

<fn_name> ::=
       <letter>[<letter> | <digit> | _ ] ...

<arg_list> ::=
       <arg> | <arg> <arg_list>

<arg> ::=
       <expr_or_attr> | <literal_constant> | .V.

<arith_expr> ::=
       <operand> <arith_op> <operand> |
       (<arith_expr>) <arith_op> <operand> |
       <operand> <arith_op> (<arith_expr>)

<arith_op> ::=
       + | - | * | /

<operand> ::=
       <tuple_attribute> | <function> | <literal_constant> | .V.

<bool_op> ::=
       & | |

<rel_op> ::=
       = | ^= | < | > | <= | >=

<letter> ::=
       A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z|
       a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z

<digit> ::=
       0|1|2|3|4|5|6|7|8|9

<argument_substitution> ::=
       .V.|.v.|.X.|.x.

<literal_constant> ::=
       <bit-string_constant> | <character-string_constant> |
       <arithmetic_constant>
```

NOTES: A <tuple_variable> is a user-specified variable that need not appear in the data model or data submodel and need not be declared in the calling program. In the <range_expression> each <tuple_variable> is associated with a <relation>. Hence, the <range_expression> defines the data base subset from which the desired data elements are to be selected. More than one <tuple_variable> may be associated with one <relation>.

The -no_optimize (-no_ot) option in the range expression causes MRDS to select tuples from the tuple variables in the order in which they are defined, (i.e., the order of range definition). No attempt will be made to find a quicker search order (see Section 13 "Performance Considerations").

The -print_search_order (-pso) option in the range expression causes MRDS to print, via the user_output switch, the order in which types from each tuple variable are selected, the type of access mechanism used to select those tuples, and the estimated number of tuples selected. (See Section 13 "Performance Considerations".)

The <set_op>s -inter, -union, and -differ correspond to the set operators intersection, union, and difference as defined in Appendix C.

Specification of a <tuple_attribute> in the <tuple_expression> results in the selection of only the specified attribute value within the designated tuple. A "*" suffix on a tuple attribute indicates that this attribute is to be a key attribute for a relation defined by define_temp_rel and is not otherwise allowed. Specification of a <tuple_variable> within the <tuple_expression> results in the selection of all attribute values in the designated tuple.

For modifications or deletions, a <select_set> must consist of one <alpha_expression>, with only one relation specified in the select clause. The operation applies to all tuples selected.

The order of evaluation of <term>s within a <qualifier>, of <operand>s within an <arith_expr>, and of <alpha_expression>s within a <select_set> is governed by the parentheses.

A <selection_expression> consisting of a <current_expression> indicates that the most recently selected occurrence of the <tuple_variable> specified in the <tuple_item> is to be selected again. The specification of a <current_expression> is valid only if a <selection_expression> consisting of a <select_set> has previously been specified. The <select_set> in this case must consist of one <alpha_expression>. The <tuple_item>s must all be in the same <tuple_variable> and that <tuple_variable> must not have been quantified in the <selection_expression>. This feature is useful primarily in calls to dsl_$delete and dsl_$modify in conjunction with "-another" retrievals.

If the <tuple_expression> contains the -dup option then only retrieve operations are permitted and duplicate items retrieved from the data base as specified in the <tuple_expression> are returned to the caller. If -dup is not specified, then only one instance of duplicate selected items is returned. The -dup option is not allowed with set operations. A retrieve operation with the -dup option may be significantly faster. However, it is important to note that there are two ways in which duplicates can occur. First, a duplicate may occur through restricted views of relations, called a projection, yielding multiple values that are duplicates of the retrieved tuples. A second type of duplication may occur when the selection expression can select the same tuple multiple times, as when the logical or (|) is used in the where clause. The -dup option permits both of these types of duplication to be seen by the user.

".V." or ".v." is an argument indicating that the values to be substituted into the <selection_expression> are to be selected from the arguments immediately following the <selection_expression> parameter. The specification of a ".V." argument as a <relation> within a <range_definition> indicates that the temporary relation with the index corresponding to the value passed via the selection value parameter is to be incorporated into the range of the associated <tuple_variable>. There must exist one selection value for each temporary relation specified in the corresponding <selection_expression>. Temporary relations are defined by calls to dsl_$define_temp_rel. Only a temporary relation index, not a relation name, may be used as a substitution value for a ".V." argument in a range clause. When a ".V." argument is found in a where clause, values are substituted for the ".V." argument from arguments following the selection expression in the argument list of the dsl_ call. These values are interpreted as literal constants by MRDS and not as relation or attribute names.

".X." or ".x." is an argument similar to ".V." (above), but it can only be used when compiling the selection expression in the call to dsl_$compile. It is used to specify an argument that is not known at the time of compilation.

If the <selection_expression> specified for a call to dsl_$retrieve results in the selection of more than one tuple, only one is returned to the caller. If the <selection_expression> consists of "-compiled" and is supplied with a selection expression index, the compiled selection expression is returned for use in the call operation for any dsl_ entry (except dsl_$compile). This compiled selection expression does not destroy the availability of any previously compiled selection expressions. However, the caller may

individually retrieve the other tuples by successive calls to dsl_$retrieve with a <selection_expression> consisting of "-another". A call to any dsl_ entry with a <selection_expression> consisting of a <select_set> terminates the availability of any previously selected tuples.

If no <qualifier_expression> is specified, all tuples in the specified range are selected.

An <fn_name> may be the name of a built-in function or of a user-defined function. Refer to Section 5 for a discussion of those functions provided as a standard part of MRDS and to the dsl_$declare subroutine in this section for information on user-defined functions.

All <tuple_attribute>s within an <expr> must have the same <tuple_variable>.

Items within a <selection_expression> are delimited by blanks, new-lines, and horizontal-tab characters not contained in quoted strings.

Where Clause Comparisons

When comparisons between attributes are specified in the where clause, the following conventions are followed (at least one of the attributes in the pair may not be a key or index).

- If either attribute is a complex number, the comparison takes place as a complex number comparison, after any necessary conversion to complex float decimal (59) numbers. Note that only "=", and "^=", are valid in this case.
- If both attributes are bit or both are character, then the comparison is done as bit or character, respectively.
- If one attribute is declared bit and the other character, the comparison is done as a character compare, after first converting the bit value.
- If either attribute is a real number and neither is complex, the comparison takes place as a real number after any necessary conversion. Real number comparisons are done as float decimal (59) number compares, unless both attributes are declared fixed binary with equal scale or float binary. In these cases, the comparisons are done as fixed bin (71) or float bin (63) compares, respectively.

If both attributes are a key head, total key, or secondary index, or if multiple attributes making up the total primary key are involved in the comparison with the condition that the attributes are of differing data types, then no convention is followed. The comparison takes place as the data type of whichever attribute the search mechanism uses for a key search. It is recommended that this case be avoided, as possible conversion errors may result.

The use of "=" with floating-point numbers is not recommended, as the comparisons may not be meaningful due to roundoff error.

The most efficient number comparisons, in terms of time and space, are with both attributes declared fixed bin or both declared float bin, preferably aligned. Both attributes declared bit or both attributes declared character are also efficient comparisons.

Examples of Selection Mechanisms

The sample data base to which the following examples apply consists of four relations, each shown with their attributes in parentheses and their key attributes followed with an "*".

```
supplier (supplier_no* supplier_name location)
part (part_no* part_name color weight quant_on_hand)
project (proj_no* proj_name manager_no)
supply (supplier_no* part_no* proj_no* ship_date* quantity)
```

1. Find all the part numbers of parts being supplied.

```
"-range (s supply)
 -select s.part_no"
```

2. Find the part numbers, names, and quantities on hand where the quantity on hand is less than 25.

```
"-range (p part)
 -select p.part_no p.part_name p.quant_on_hand
 -where p.quant_on_hand < 25"
```

3. Find the supplier numbers of those suppliers who supply the part with the part number 3.

```
"-range (z supply)
 -select z.supplier_no
 -where z.part_no = 3"
```

4. Find the supplier names of those suppliers who supply the part with the part number 3.

```
"-range (s supplier) (z supply)
 -select s.supplier_name
 -where ((s.supplier_no = z.supplier_no) &
        (z.part_no = 3))"
```

5. Find the supplier numbers of those suppliers who have the same location as supplier Jones.

```
"-range (s supplier) (t supplier)
 -select s.supplier_no
 -where ((t.supplier_name = ""Jones"") &
        (t.location = s.location))"
```

6. For each project, find the project number, project name, and supplier location for all suppliers who supply that project.

```
"-range (p project) (s supplier) (z supply)
 -select p.proj_no p.proj_name s.location
 -where ((p.proj_no = z.proj_no) &
        (z.supplier_no = s.supplier_no))"
```

Name: dsl_

This subroutine supplies entry points for the functions required in opening, manipulating, and closing a data base. (Refer to "Obsolete Interfaces," Section 10, for additional, but obsoleted dsl_ entries.

Usage of the dsl_ subroutine is explained below under separate headings for each designated entry.

NOTES: The sub_error_ condition is signaled for some errors to provide further information. It is suggested that an on unit (refer to the PL/I Reference Manual) be established to trap this error after program development work is complete.

The arg_error condition is signaled for cases where the error code argument cannot be obtained.

When arguments for data are expected, as in dsl_$retrieve, a structure, like the output from create_mrds_dm_include, may be used in place of separate arguments. However, only one structure per call is allowed and there is a loss in efficiency.

The following is a summary of dsl_ entries.

close
closes the specified open data bases.

close_all
closes all data bases currently open in the user's process.

compile
compiles (or pre-translates) a selection expression for later use in the current process.

declare
makes a user-defined function known to MRDS.

define_temp_rel
defines, redefines or deletes a temporary relation that can be accessed by the current process. The only functions which can be accomplished using a temporary relation are "retrieve" and "define_temp_rel."

delete
specifies that the selected data is to be deleted from the data base.

dl_scope
deletes all or a portion of the current scope of access.

dl_scope_all
deletes all of the current scope of access.

get_attribute_list
returns attribute descriptions and access capabilities for all attributes in the user's view of a given relation.

get_opening_temp_dir
returns the directory pathname used for temporary storage in a particular data base opening.

get_path_info
: returns information about a relative pathname. This includes the MRDS model/submodel absolute path, version, and creation information if the path refers to a MRDS model or submodel.

get_population
: returns the current number of tuples stored in a permanent or temporary relation.

get_relation_list
: returns a list of all relations in the view of a given opening, plus access capabilities for each relation.

get_scope
: returns the scope currently set on a given relation.

get_temp_dir
: returns the directory pathname used for temporary storage on the next call to dsl_$open.

list_openings
: returns a list of all currently open data bases.

modify
: specifies that the selected portion of the data base is to be modified.

open
: opens the specified data bases or data submodels for processing.

retrieve
: displays the selected data specified by the selection expression.

set_scope
: defines the current scope of access for a relation.

set_scope_all
: defines the current scope of access for all relations.

set_temp_dir
: sets the directory used for temporary storage on the next call to dsl_$open.

store
: adds a new tuple to the selected relation.

Entry: dsl_$close

This entry causes the specified data bases to be closed and made unavailable for processing.

Usage

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

call dsl_$close (data_base_index1, ... , data_base_indexn, code);
```

where:

1. data_base_indexi (Input) (fixed bin(35))
   is the integer returned by dsl_$open that designates the currently open data bases that are to be closed.

2. code (Output) (fixed bin(35))
   is a standard status code.

Entry: dsl_$close_all

This entry closes all data bases that are currently open in the user's process.

Usage

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

call dsl_$close_all (code);
```

where code (output) (fixed bin(35)) is the standard status code and is 0 if all data bases are successfully closed or if no data bases are open.

Entry: dsl_$compile

This entry compiles (or pre-translates) a selection expression for later use in the current process, for retrieval, modify, delete, and define_temp_rel operations. A previously compiled selection expression can be deleted or redefined through this entrypoint.

A selection expression can be compiled at any time in the life of an open data base and saved for future use in that opening.

Usage

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

call dsl_$compile (data_base_index, selection_expression, se_index,
     se_value1, ..., se_valuen, code);
```

where:

1. data_base_index (Input) (fixed bin(35))
   is the index returned by dsl_$open to designate the data base.

2. selection_expression (Input) (char (*))
   is a character string as defined at the beginning of this section (see "Formal Definition of the Selection Expression"). It may contain .V. argument substitution characters in all normal places. These are filled in at the time the selection expression is compiled. Argument substitution characters of the form .X. may be used in all places in the where clause where .V. is appropriate, except functions and expressions, to specify that this value is to be filled in at the time that the selection expression is used.

3. se_index (Input/Output) (fixed bin(35))
   is an integer used to identify a compiled selection expression. If the se_index is 0 (on input), a new compiled selection expression is defined and the index for the newly compiled selection expression is returned. If the se_index is greater than zero (on input) and a compiled selection expression with that index is found, it is redefined to the new selection expression. If the se_index is less than zero (on input) and a compiled selection expression with that index is found, it is deleted and the selection expression is ignored.

4. se_valuei (Input)
   is a selection expression value for each control code (designated by .V.) appearing in the selection expression. These must be specified so as to correspond in order and quantity with the control codes specified in the selection expression.

5. code (Output (fixed bin(35))
   is a standard MRDS status code. A value of 0 indicates that no error occurred.

Note

Any .V. argument substitution characters supplied in the selection expression must have matching arguments supplied in the call to dsl_$compile. They are then considered to be constant and cannot be changed later. Any .X. argument substitution characters supplied in the selection expression must have matching arguments supplied in the call that references the compiled selection expression, not the call to dsl_$compile. The arguments supplied to satisfy a .X. must have the same data type as that of the data base attribute it is being compared to.

dsl_

MRDS selection expressions are optimized at the time they are compiled. The search method chosen is highly data-dependent. When using compiled selection expressions in a situation where the data is changing rapidly, the optimization chosen at compilation time may not be the same optimization which would be appropriate at execution time.

dsl_

Entry: dsl_$declare

This entry makes a user-defined function known to MRDS while processing the specified data base. After it is declared, a user-defined function may be used exactly as a MRDS built-in function. If a user-defined function has the same name as a built-in function, the user-defined function is referenced.

Usage

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

call dsl_$declare (db_index, fn_name, code);
```

where:

1. db_index (Input)
   is the index returned by dsl_$open that designates the data base.

2. fn_name (Input)
   is the name of the function being declared.

3. code (Output)
   is a standard status code.

Notes

Built-in functions are provided as a standard part of MRDS and need not be declared. These functions are described in Section 5.

User-defined functions may be written in PL/I, COBOL, or FORTRAN. MRDS generates a call that is equivalent to:

```
return_val = fn_name$fn_name (arg1 ... argn);
```

Restrictions on arguments to user-defined functions are:

1. No star (*) extents are permitted in the declarations for return_val or argi.
2. Data types are restricted to those data types permitted in a MRDS data base (i.e., pointers, entries, labels, structures, offsets, and arrays are not allowed).

Example

Declare the user-defined function "state":

```
call dsl_$declare (db_index, "state", code);
```

See "Writing Nonstandard Functions" in Section 5.

Entry: dsl_$define_temp_rel

This entry allows the user to explicitly create, delete, or redefine a temporary relation that can be used by the current process for retrieval operations in the same manner as any predefined permanent data base relations.

The only operations that can be performed on a temporary relation are the "define_temp_rel", "retrieve", and "get_population". After a temporary relation is defined, it is referenced by specifying a ".V." argument in the range clause and supplying the appropriate rel_index in the dsl_ call argument list. A temporary relation cannot be used in the -select clause except for the dsl_$retrieve call.

Usage

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

call dsl_$define_temp_rel (data_base_index, selection_expression,
     se_index, se_value1, ... , se_index, se_valuen, rel_index, code);
```

where:

1. data_base_index (Input) (fixed bin(35))
   is the index returned by dsl_$open that designates the data base.

2. selection_expression (Input) (char(*))
   is a character string (see "Examples of Selection Mechanisms" above) as defined at the beginning of this section, with at least one * in the -select clause to define the temporary relation key. The attribute names given in the select clause must be unique. This character string may be a constant or a variable declared either character varying or non-varying.

3. se_index (Input) (fixed bin(35))
   is an integer used to refer to a compiled selection expression. It is required only if the selection expression is "-compiled".

4. se_valuei (Input)
   is a selection expression value for each argument substitution (designated by .V. or .X.) appearing in the <selection_expression>, including temporary relation (rel_index) designations. These must be specified so as to correspond in order and quantity with the argument substitution specified in the <selection_expression). If the selection expression is "-compiled", then the selection expression value is substituted for the .X. value in the where clause that has to be satisfied. These values are supplied in the order in which they occur in the selection expression used in the call to dsl_$compile. If the specified data type does not equal the attribute data type, the value mrds_error_$inv_data_type is returned in the code.

5. rel_index (Input/Output) (fixed binary(35))
   is an integer. If rel_index is 0 on input, a new temporary relation is defined and the index for the newly created temporary relation is returned in rel_index. If rel_index is greater than 0 on input and if a temporary relation possessing this index is already in existence, that temporary relation is redefined. If rel_index is less than zero and a temporary relation with that rel_index exists, then that temporary relation is deleted and the selection expression is ignored.

6. code (Output) (fixed bin(36))
is a standard status code. A value of 0 indicates that no error occurred.

Notes

If a duplicate of the temporary relation key is found while creating the temporary relation, it is ignored (i.e., not stored) without warning.

If no data satisfied the selection expression, then an unpopulated temporary relation is created. The population can be determined by a call to dsl_$get_population.

For shared openings, relations specified in the range clause must have read_attr scope set.

For attribute level security, attributes specified in the select and where clauses must have read_attr access.

Entry: dsl_$delete

This entry allows the user to delete one or more tuples from the same relation of an opened data base. The user must have read-write permission to the relation. All attributes in the relation must be specified as being selected and, if the data base is being referenced by means of a data submodel, all attributes of the relation must be defined in the submodel. All selected tuples are deleted.

Usage

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

call dsl_$delete (data_base_index, selection_expression, se_index,
     se_value1, ... , se_valuen, code);
```

where:

1. data_base_index (Input) (fixed bin(35))
   is the index returned by dsl_$open that designates the data base.

2. selection_expression (Input) (char(*))
   is a character string (see "Selection Mechanism") as defined at the beginning of this section. However, the select clause must specify all attributes in the relation. This character string may be a constant or a variable declared character varying or non-varying.

3. se_index (Input) (fixed bin(35))
   is an integer used to refer to a compiled selection expression. It is required only if the selection expression is "-compiled".

4. se_valuei (Input)
   is a selection expression value for each argument substitution (designated by .V. or .X.) appearing in the <selection_expression>, including temporary relation (rel_index) designations. These must be specified so as to correspond in order and quantity with the argument substitution specified in the <selection_expression). If the selection expression is "-compiled", then the selection expression value is substituted for the .X. value in the where clause that has to be satisfied. These values are supplied in the order in which they occur in the selection expression used in the call to dsl_$compile. If the specified data type does not equal the attribute data type, the value mrds_error_$inv_data_type is returned in the code.

5. code (Output) (fixed bin(35))
   is a standard status code. A value of 0 indicates that no error occurred. A value corresponding to mrds_error_$tuple_not_found indicates that no error occurred and that no data satisfied the selection expression.

Notes

For shared openings, the relation must have delete_tuple permit scope set.

For attribute level security, the relation must have delete_tuple access and any attributes specified in the where clause must have read_attr access.

Entry: dsl_$dl_scope

This entry deletes all or part of a previously specified scope from the user's current scope.

Usage

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

call dsl_$dl_scope (db_index, rel_name1, permit_ops1, prevent_ops1, ... .
     rel_namen, permit_opsn, prevent_opsn, code);
```

where:

1. db_index (Input) (fixed bin(35))
   is the index returned by dsl_$open that designates the data base.

2. rel_namei (Input) (char(*))
   is the name of the relation(s) to be included in the scope.

3. permit_opsi (Input) (fixed bin)
   the sum of the codes for the operations no longer granted to the user's process on the specified relation (as defined in set_scope). (See "Note" below for a description of appropriate codes.)

4. prevent_opsi (Input) (fixed bin)
   the sum of the codes for the operations no longer denied to other processes on the specified relation (as defined in set_scope). (See "Note" below for a description of appropriate codes.)

5. code (Output) (fixed bin(35))
   is a standard status code.

Note

Scope codes for operations to be prevented or permitted are as follows:

Scope Code	Operation
0	null
1	read_attr or read
2	append_tuple or store
4	delete_tuple or delete
8	modify_attr or modify

Current scope settings can be determined by a call to dsl_$get_scope.

See mrds_call delete_scope function examples.

Entry: dsl_$dl_scope_all

This entry deletes all remaining scope tuples from the user's current scope.

Usage

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

call dsl_$dl_scope_all (db_index, code);
```

where:

1. db_index (Input)
   is the index returned by dsl_$open that designates the data base.

2. code (Output)
   is a standard status code. It is 0 if no scope is set prior to the call to dsl_$dl_scope_all.

Entry: dsl_$get_attribute_list

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

Usage

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

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

where:

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

2. relation_name (Input) (char(*))
   is the name of the relation in the user's view for which the attribute information is desired.

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

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

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

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

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

   error_table_$badcall
   if the area_ptr was null.

   error_table_$unimplemented_version
   if the structure_version supplied is unknown.

   mrds_error_$invalid_db_index
   if the db_index given does not refer to a data base open in this process.

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

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

mrds_error_$version_not_supported
if the data base referenced is not version 4.

Notes

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

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

where:

1. version
   is the version number of this structure and should be set by the caller to mrds_attribute_list_structure_version.

2. access_info_version
   is the version of the MRDS access modes returned in the attribute information. Version 4 refers to version 4 data bases without attribute level security, using r-w system ACLs. Version 5 refers to secured version 4 data bases with attribute level security using read_attr (r) and modify_attr (m) attribute access modes.

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

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

5. mbz1
   is reserved for future use.

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

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

8. domain_name
   is the name of the underlying domain for this attribute. If the data base is secured and the caller is not a DBA, then this field is blank.

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

10. system_acl
    is the Multics ACL on this attribute from the modes r-w.

11. mrds_access
    is the MRDS access mode for this attribute. See the access_info_version description for possible values for various versions of MRDS access control.

12. effective_access
    is the result of applying both system ACLs and MRDS access to this attribute, using MRDS access values for the effect.

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

14. mbz2
    is reserved for future use.

The only structure version currently available is 1. This entry only works for version 4 data bases.

The variables mrds_attribute_list_num_attrs_init, mrds_attribute_list_ptr, and mrds_attribute_list_structure_version are also declared in the mrds_attribute_list include file.

Entry: dsl_$get_opening_temp_dir

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

Usage

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

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

where:

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

2. error_code (Output) (fixed bin(35))
   is the standard status code. If the supplied db_index does not refer to a currently open data base in the user's process then it will be mrds_error_$invalid_db_index.

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

Notes

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

Entry: dsl_$get_path_info

This entry returns information about a supplied pathname. It indicates whether or not the path refers to a MRDS data base model or submodel, and if so, the version number and details about its creation. This entry replaces dsl_$get_db_version, which is obsolete (see Section 10).

Usage

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

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

where:

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

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

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

4. mrds_path_info_ptr (Output) (pointer)
   is the pointer to the path information structure that is returned, which is described in the Notes below.

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

   error_table_$area_too_small
   if the supplied area could not hold the path information.

   error_table_$badcall
   if the area_ptr was null.

   error_table_$unimplemented_version
   if the supplied structure version is unknown.

   mrds_error_$no_model_access
   if the user does not have "r" access to the db_model segment under the data base.

   mrds_error_$no_model_submodel
   if the path does not refer to a MRDS data base model or submodel.

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

Notes

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

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

where:

1. version
   is the version number of this structure and should be set by the caller to mrds_path_info_structure_version.

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

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

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

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

6. mbz1
   is reserved for future use.

7. mrds_version
   is the version number of the MRDS model or submodel that was found. The latest version data base model is 4 and for submodels it is 5.

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

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

10. mbz2
    is reserved for future use.

The only structure version currently available is 1. The variables mrds_path_info_ptr and mrds_path_info_structure_version are also declared in the mrds_path_info include file.

Entry: dsl_$get_population

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

Usage

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

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

where:

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

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

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

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

   mrds_error_$invalid_db_index
   if the given db_index does not refer to a model or submodel opening of a data base in the user's process.

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

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

Notes

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

*

Entry: dsl_$get_relation_list

This entry returns information about all the relations in the specified opening view.

Usage

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

call dsl_$get_relation_list (db_index, area_ptr,
     structure_version, mrds_relation_list_ptr,
     error_code);
```

where:

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

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

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

4. mrds_relation_list_ptr (Output) (pointer)
   is a pointer to the relation information structure that has been allocated and is described in the Notes below.

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

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

   error_table_$badcall
   if the area_ptr was null.

   error_table_$unimplmented_version
   if the supplied structure version is unknown.

   mrds_error_$invalid_db_index
   if the db_index given does not refer to a data base open in this process.

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

   mrds_error_$version_not_supported
   if the data base referenced is not version 4.

Notes

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

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

where:

1. version
   is the version number for this structure and should be set by the caller to mrds_relation_list_structure_version.

2. access_info_version
   is the version number of the access information being returned. Version 4 is for version 4 data bases without attribute level security using Multics ACLs from r-w. Version 5 is for secured version 4 data bases with attribute level security using the MRDS relation access modes of append_tuple (a) and delete_tuple (d).

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

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

5. mbz1
   is reserved for future use.

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

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

8. system_acl
   is the Multics ACL on the relation data from the modes r-w.

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

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

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

12. mbz2
    is reserved for future use.

Currently, the only structure version available is 1. The variables mrds_relation_list_num_rels_init, mrds_relation_list_ptr, and mrds_relation_list_structure_version are also declared in the mrds_relation_list include file.

Entry: dsl_$get_scope

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

Usage

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

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

where:

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

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

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

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

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

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

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

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

Notes

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

Scope Code	Operation
0	null
1	read_attr or read
2	append_tuple or store
4	delete_tuple or delete
8	modify_attr or modify

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

Entry: dsl_$get_temp_dir

This entry returns the pathname of the directory that is used for temporary storage upon the next call to dsl_$open.

Usage

```
declare dsl_$get_temp_dir entry () returns (char(168));

path = dsl_$get_temp_dir ();
```

where path (Output) (char(168)) is the absolute pathname of the directory to be used for temporary storage on the next call to open.

* Notes

See dsl_$set_temp_dir and the commands display_mrds_temp_dir and set_mrds_temp_dir.

To obtain the temporary storage directory for a particular opening, call dsl_$get_opening_temp_dir.

Entry: dsl_$list_openings

This entry returns information about all openings of MRDS data bases in the user's process. This entry replaces dsl_$list_dbs, which is obsolete (see Section 10).

Usage

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

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

where:

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

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

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

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

   error_table_$area_too_small
   if the supplied area could not hold the opening information.

   error_table_$badcall
   if the area_ptr was null.

   error_table_$unimplemented_version
   if the given structure_version is unknown.

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

Notes

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

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

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

where:

1. version
   is the version number of this structure and should be set by the caller to mrds_database_openings_structure_version.

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

3. mbz1
   is reserved for future use.

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

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

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

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

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

9. mbz2
   is reserved for future use.

Currently, the only structure version available is 1.

The following variables are also declared in the mrds_database_openings include file.

```
mrds_database_openings_ptr
mrds_database_openings_num_open_init
mrds_database_openings_structure_version
```

Entry: dsl_$modify

This entry allows the user to modify attribute values contained in the tuples of one relation in the data base. The modification of a key attribute is not allowed. The user must have read_write permission to the relation. All selected tuples are modified.

Usage

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

call dsl_$modify (data_base_index, selection_expression, se_index,
     se_value1, ... , se_valuen, modified_value1, ... , modified_valuen,
     code);
```

where:

1. data_base_index (Input) (fixed bin(35))
   is the index returned by dsl_$open that designates the data base.

2. selection_expression (Input) (char(*))
   is a character string (see "Examples of Selection Mechanisms") as defined at the beginning of this section. The select clause can only specify attributes from one relation. This character string may be a constant or a variable declared character varying or non-varying.

3. se_index (Input) (fixed bin(35))
   is an integer used to refer to a compiled selection expression. It is required only if the selection expression is "-compiled".

4. se_valuei (Input)
   is a selection expression value for each argument substitution (designated by .V. or .X.) appearing in the <selection_expression>, including temporary relation (rel_index) designations. These must be specified so as to correspond in order and quantity with the argument substitution specified in the <selection_expression). If the selection expression is "-compiled", then the selection expression value is substituted for the .X. value in the where clause that has to be satisfied. These values are supplied in the order in which they occur in the selection expression used in the call to dsl_$compile. If the specified data type does not equal the attribute data type, the value mrds_error_$inv_data_type is returned in the code.

5. modified_valuei (Input)
   is a modified tuple attribute value that is to replace the current such value in the data base. There must be a one-to-one correspondence between these values and the tuple items specified in the selection expression. If a structure is used for modified tuple attribute values, only one structure may be used. Only data types supported by assign_ may be used for modified tuple attribute values.

6. code (Output) (fixed bin(35))
is a standard status code. A value of 0 indicates that no error occurred. A value of mrds_error_$tuple_not_found indicates that no error occurred and that no data satisfied the selection expression.

Notes

For shared openings, the relation must have modify_attr permit scope set.

For attribute level security, the attributes specified in the select and where clauses must have read_attr access. In addition, the attributes specified in the select clause must have modify_attr access.

This page intentionally left blank.

Entry: dsl_$open

This entry causes the specified data bases to be opened for processing in the designated modes. For each opened data base, an index that is to be used to specify that data base in future MRDS calls is returned. If one or more of the data bases specified cannot be opened for any reason, none of the others are opened.

Usage

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

call dsl_$open (path1, data_base_index1, mode1, ... ,
     pathn, data_base_indexn, moden, code);
```

where:

1. pathi (Input) (char(*))
   is a character string containing the absolute or relative pathname of the data submodel (or the data base) with or without a suffix defining the relevant portion of the data base. If the path of the data base itself is specified, the data model is used in place of the data submodel.

2. data_base_indexi (Output) (fixed bin(35))
   is an integer that is to be used in subsequent MRDS calls to specify the corresponding data base designated in this opening.

3. modei (Input) (fixed bin(35))
   is an integer (1,2,3,or 4) indicating the usage mode for which the data base is to be opened.

   1 specifies that this is a shared opening, requiring the setting of concurrency control protection via scope requests by the set_scope function. The maximum permit scope that can be set with this opening mode is read_attr.

   2 specifies that this is a shared opening, requiring the setting of concurrency control protection via scope requests by the set_scope function. Any scope can be set with this opening mode.

   3 specifies that this is an unshared opening in the sense that all update operations are prevented against any relations in this view of the data base. 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 permit of read_attr and prevents of modify_attr, append_tuple, and delete_tuple on these relations. Other data base openers are allowed to set read_attr scope and to do retrievals in these relations.

   4 specifies that this is an unshared opening. No scope setting is necessary with this opening mode. No other data base openers are allowed to set any scope or any relation in this view of the data base. 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. Only one opening with this mode is allowed if the set of relations in this view overlaps the relations in another opener's view.

4. code (Output) (fixed bin(35))
is a standard status code.

Notes

Open modes 1 and 2 require subsequent calls to the dsl_ entry set_scope. Also see Appendix F for the include file mrds_open_modes_.incl.pl1.

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

If the data base being opened has been secured, then the view_path must refer to a submodel that resides in the "secure.submodels" directory under the data base directory if the user is not a DBA. These must be version 5 submodels if attribute level security is to be provided. See secure_mrds_db and Section 7, "Security".

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

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

A maximum of 128 openings of the same or different data bases is allowed. Only 64 of these openings can be version 3 or earlier data bases.

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 data base concurrency control segment. Unshared openings require that, for any relation appearing in the view, the multisegment file containing the data must have "r" ACL for exclusive_retrieval or "rw" ACL for exclusive_update opening mode. For attribute level security, exclusive_retrieval mode requires read_attr on some attribute in each relation in the opening view and exclusive_update mode requires one of append_tuple on the relation, delete_tuple on the relation, or modify_attr on some attribute in the relation, for each of the relations in the opening view.

See the examples for the mrds_call function open.

Entry: dsl_$retrieve

This entry allows the user to retrieve selected attribute values from the data base. The user must have read permission to the referenced relations. One tuple per call is returned.

Usage

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

call dsl_$retrieve (data_base_index, selection_expression, se_index,
     se_value1, ... , se_valuen, value1, ... , valuen, code);
```

where:

1. data_base_index (Input) (fixed bin(35))
   is the index returned by dsl_$open that designates the data base.

2. selection_expression (Input) (char(*))
   is a character string (see "Formal Definition of the Selection Expression" in this section). This character string may be a constant or a variable declared character varying or nonvarying. If the expression results in the selection of identical tuples, only one copy is returned unless the -dup option is specified. However, all tuples selected remain available for retrieval with additional calls to dsl_$retrieve with a <selection_expression> consisting of "-another". They cease to be available whenever any dsl_ entry is called with a <selection_expression> consisting of an <alpha_expression>. The selection expression "-another" does not return duplicate tuples unless the -dup option was specified in the original <alpha_expression>. The -dup option cannot be used with set operations. The range clause may have a ".V." for substitution of a temporary relation's rel_index.

3. se_index (Input) (fixed bin(35))
   is an integer used to refer to a compiled selection expression. It is required only if the selection expression is "-compiled".

4. se_valuei (Input)
   is a selection expression value for each argument substitution (designated by .V. or .X.) appearing in the <selection_expression>, including temporary relation (rel_index) designations. These must be specified so as to correspond in order and quantity with the argument substitution specified in the <selection_expression). If the selection expression is "-compiled", then the selection expression value is substituted for the .X. value in the where clause that has to be satisfied. These values are supplied in the order in which they occur in the selection expression used in the call to dsl_$compile. If the specified data type does not equal the attribute data type, the value mrds_error_$inv_data_type is returned in the code.

5. valuei (Output)
   is a retrieved attribute value. The value may be a structure (only one regardless of the number of relations) or a list of individual values, the items of which must correspond in order and quantity with the tuple items specified in the <selection_expression>. If an entire tuple is retrieved by specifying only the tuple_value in the select clause, then a value must be specified for every attribute of the corresponding relation as defined in the data submodel or in the data model, whichever is being

used. If data conversion is required, only data types supported by assign_ may be used.

6. code (Output) (fixed bin(35))
is a standard status code. A value of 0 indicates that no error occurred and that one occurrence of the specified data has been successfully retrieved. A value of mrds_error_$tuple_not_found indicates that no error occurred and that no data satisfied the selection expression.

Notes

For shared openings, the referenced relations must have read_attr permit scope set.

For attribute level security, attributes referenced in the select and where clauses must have read_attr access.

Entry: dsl_$set_scope

This entry defines the user's current scope of access to the data base for shared modes of openings. Before a user can access the data base in shared mode, a scope of access must be declared, consisting of a set of scope tuples. If this scope does not conflict with any other currently existing scope (of other processes), it is accepted. Otherwise, the user's request is placed in a queue and is processed as soon as the requested resources become available. If the specified wait time is exceeded before the request can be processed, an error code is returned. Once the scope has been accepted, only operations permitted by the scope may be performed. As time progresses in the current process, individual scope tuples may be removed as they are no longer needed by invoking dsl_$dl_scope. However, new tuples may not be added to the current scope until all current scope has been deleted. This rule avoids potential deadlock problems within the data base manager.

Usage

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

call dsl_$set_scope (db_index, rel_name1, permit_ops1, prevent_ops1, ....
        rel_namen, permit_opsn, prevent_opsn, wait_sec, code);
```

where:

1. db_index (Input) (fixed bin (35))
   is the index returned by dsl_$open that designates the data base.

2. rel_namei (Input) (char(*))
   is the name of the relation to be included in the scope.

3. permit_opsi (Input) (fixed bin)
   is an integer consisting of the scope code which indicates the operations the user may perform on the relation.

4. prevent_opsi (Input) (fixed bin)
   is an integer consisting of the scope code which indicates the operations that other users may not perform on the relation.

5. wait_sec (Input) (fixed bin (35))
   specifies the maximum number of seconds to wait for the scope request to be honored (there is no anticipated maximum). This argument is optional and if not provided by the user, the default is 30 seconds.

6. code (Output) (fixed bin (35))
   is a standard status code. The code is 0 if set_scope is successful or is mrds_error_$db_busy if the data base is busy.

Note

Codes for operations to be prevented or permitted are:

Scope Code	Operation
0	null
1	read_attr or read
2	append_tuple or store
4	delete_tuple or delete
8	modify_attr or modify

It is not necessary to set scope on temporary relations or on relations in a data base which was opened with an exclusive opening mode. (See Appendix F for the include file mrds_new_scope_modes.incl.pl1.)

Access requirements on the relation(s) for which scope is being set in terms of Multics ACLs 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

Example

The following example shows the appropriate "call" to define scope on relation "employee" such that the user's process has retrieve access to the relation while all other processes are prevented from stores, modifies, and deletes (as might be necessary in doing a totalling operation within a relation). If the request cannot be honored within 60 seconds, a mrds_error_$db_busy code is issued to the calling program.

```
call dsl_$set_scope (db_index, "employee", 1, 14, 60, code);
```

Also see the mrds_call set_scope function examples.

Entry: dsl_$set_scope_all

This entry provides a means of setting a scope on all relations defined in the user's view without the need to name each relation. Identical permit operations and prevent operations are applied to all the relations in the user's view.

*

Usage

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

call dsl_$set_scope_all (db_index, permit_ops, prevent_ops, wait_sec, code);
```

where:

1. db_index (Input) (fixed bin(35))
   is the index returned by dsl_$open that designates the data base.

2. permit_ops (Input) (fixed bin)
   is the scope code which indicates the operations the user may perform on the relation.

3. prevent_ops (Input) (fixed bin)
   is the scope code which indicates the operations that other users may not perform on the relation.

4. wait_sec (Input) (fixed bin(35))
   specifies the maximum number of seconds to wait for the scope request to be honored (there is no anticipated maximum). This argument is optional and, if not provided by the user, the default is 30 seconds.

5. code (Output) (fixed bin(35))
   is a standard status code.

Note

Scope codes for operations to be prevented or permitted are:

Scope Code	Operation
0	null
1	read_attr or read
2	append_tuple or store
4	delete_tuple or delete
8	modify_attr or modify

See the mrds_call set_scope_all function examples.

See dsl_$set_scope for access requirements.

Entry: dsl_$set_temp_dir

This entry sets the directory that is used for temporary storage on the next call to dsl_$open. This temporary directory has a default of process directory. Therefore, this entry need never be called unless a record quota overflow occurs on the process directory, as might happen in opening a data base with a large number of relations, or during a large retrieve or define_temp_rel operation.

Usage

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

call dsl_$set_temp_dir (path, code);
```

where:

1. path (Input) (char(*))
   is the relative or absolute pathname of the directory to be used for temporary storage on the next call to open.

2. code (Output) (fixed bin(35))
   is the standard status code and is 0 unless an error occurs.

Notes

See dsl_$set_temp_dir, dsl_$get_temp_dir, dsl_$get_opening_temp_dir, and the commands display_mrds_temp_dir and set_mrds_temp_dir.

See "Notes" under set_mrds_temp_dir command for proper use of this interface.

Entry: dsl_$store

This entry allows the user to add a tuple to a designated relation in the data base. The placement of the new tuple within the relation is determined by MRDS, based upon data model/data submodel descriptions of the data base and the value of the primary key in the new tuple. The primary key of the new tuple must be unique within the designated relation. The caller must have read-write permission to the relation. If storing through a submodel view, all attributes of the relation must be defined in the submodel.

Usage

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

call dsl_$store (data_base_index, relation_expression,
     new_value1, ... , new_valuen, code);
```

where:

1. data_base_index (Input) (fixed bin(35))
   is the index returned by dsl_$open that designates the data base.

2. relation_expression (Input) (char(*))
   indicates the relation to which the tuple is to be added, as it appears in the user's view of the data base (the data model or the data submodel). It may be the name of the relation or it may be "-another".

3. new_valuei (Input)
   is the new tuple value to be added to the relation. The entire tuple, as defined in the user's view, may be specified with one structure or a list of variables, the items of which must correspond in order and quantity with the attributes defined in the user's view.

4. code (Output) (fixed bin(35))
   is a standard status code. The value is 0 if the store was successful. If a duplicate of the primary key already exists in the data base, the code value mrds_error_$dup_store is returned and the tuple is not stored. (The name mrds_error_$duplicate_key may also be used.) If a -check_proc option exists on a domain of one of the attributes in the relation for which a tuple is being added and the check procedure returns false, then the error code, mrds_error_$dom_integ, is returned.

Notes

If the relation_expression is the name of a relation, the new tuple is added to the named relation. If the relation_expression is "-another", the new tuple is added to the relation specified in the most recent call to the dsl_$store in which the relation_expression argument consisted of a relation name. Any call to a dsl entry requiring a <selection_expression> causes the previously specified relation name to become unavailable for subsequent reference using

"-another", until it is again established via a call to dsl_$store with a relation_expression consisting of the relation name.

The use of "-another" provides an efficient means to store several tuples into a single relation via consecutive dsl_$store calls.

For shared openings, the relation must have append_tuple permit scope set.

For attribute level security, the relation must have append_tuple access and the key attributes must have read_attr access.

EXAMPLE -- OPENING, ACCESSING, AND CLOSING A DATA BASE

Assume the same sample data base used for the "Examples of Selection Mechanisms" (previously shown in this section). Also assume the following declarations have been made within the calling program.

```
dcl 1 supplier,
      2 supplier_no fixed bin,
      2 supplier_name char(32),
      2 location char(128);

dcl 1 part,
      2 part_no fixed bin,
      2 part_name char(16),
      2 color char(8),
      2 weight fixed bin,
      2 quant_on_hand fixed bin;

dcl 1 project,
      2 proj_no fixed bin,
      2 proj_name char(32),
      2 manager_no fixed bin;

dcl 1 supply,
      2 supplier_no fixed bin,
      2 part_no fixed bin,
      2 proj_no fixed bin,
      2 ship_date char(6),
      2 quantity fixed bin;
```

1. Open the data base for (nonexclusive) update.

```
call dsl_$open ("supply_data_submodel", db_index, 2, code);
```

2. Perform the following update, assuming the data base is opened for (nonexclusive) update.

   Add DELTA to the quantity on hand for the part with the part number 3.

```
call dsl_$set_scope (db_index,
     "part", 15, 1, code);

call dsl_$retrieve (db_index,
     "-range (p part)
```

```
          -select p.quant_on_hand
          -where p.part_no = 3",
         quant_on_hand, code);

    quant_on_hand = quant_on_hand + DELTA;

    call dsl_$modify (db_index,
          "-current p.quant_on_hand",
         quant_on_hand, code);

    call dsl_$dl_scope(db_index,
         "part", 15, 1, code);
```

3. Perform the following deletion, assuming the data base is opened for exclusive update.

   Delete all tuples of the supply relation involving supplier Jones and project Alpha in combination with one another.

```
    call dsl_$delete (db_index,
         "-range (z supply) (s supplier) (p project)
          -select z
          -where (((s.supplier_name = ""Jones"")
                 & (s.supplier_name = z.supplier_name))
                 & ((p.proj_name = ""Alpha"")
                 & (z.proj_name = p.proj_name)))",
         code);
```

4. Perform the following store operation.

   Add the tuple contained in NEW_PART to the part relation.

```
    call dsl_$store (db_index, "part", NEW_PART, code);
```

EXAMPLE -- MODIFICATION OF KEY ATTRIBUTES

When it is desirable to be able to use the equivalent of two different selection expressions with independent "-another" processing, two or more openings of the same data base may be necessary in order to maintain position "currency" within the data base for each selection expression.

An example of a multiple data base opening application is the modification of a key attribute, which must be done by a program such as follows (dsl_$modify does not work on key attributes). Note the use of the entire key in the dsl_$delete where clause and the use of the second opening index for the delete and store, so as not to lose retrieve selection expression currency for "-another" calls.

Not all declarations are shown. The modify_proc is a procedure that carries out the modification of the attribute value before it is stored; error_proc is a general error routine.

```
delete_select_expr = "-range (i invy) -select i
-where ((((i.senum = .V.) & (i.secode = .V.)) &
     (i.part = .V.)) & (i.divn = .V.))";
```

```
none = 0;
read = 1
update_only = 14
read_update = 15;
update = 2;
db_path = ">udd>Demo>dbmt>db7>jg>CS_III.db";

call dsl_$open (db_path, dbi_1, update, db_path,
     dbi_2, update, code);
if code ^= 0 then call error_proc();

call dsl_$set_scope_all (dbi_1, read, none, code);
if code ^= 0 then call error_proc;

call dsl_$set_scope_all (dbi_2, read_update, update_only, code);
if code ^= 0 then call error_proc;

first_time = "1"b;
do while (code = 0);

     if first_time then do;
     retrieve_select_expr =
            "-range (i invy) -select i";
       first_time = "0"b;
     end;
     else retrieve_select_expr = "-another";

     call dsl_$retrieve (dbi_1, retrieve_select_expr, invy, code);

     if code = 0 then
          call dsl_$delete (dbi_2, delete_select_expr,
               invy.senum, invy.secode, invy.part, invy.divn, code);

     invy.senum = modify_proc (invy.senum);

     if code = 0 then
          call dsl_$store (dbi_2, "invy", invy, code);
end;
if code ^= mrds_error_$tuple_not_found then
     call error_proc();
```

```
/* ***********************************************
   *                                             *
   * BEGIN CS_III.incl.pl1                       *
   *      created: 02/01/80  1439.2 mst Fri      *
   *           by: create_mrds_dm_include (2.0)  *
   *                                             *
   * Data model >udd>STL>mrds_dev>db>CS_III.db   *
   *      created: 02/01/80  1438.0 mst Fri      *
   *      version: 4                             *
   *           by: JGray.Multics.a               *
   *                                             *
   *********************************************** */

                                                                                *
dcl 1 invy aligned,
      2 senum character (8) nonvarying unaligned,                    /* Key */
      2 secode character (1) nonvarying unaligned,                   /* Key */
      2 part character (3) nonvarying unaligned,                     /* Key */
      2 divn character (3) nonvarying unaligned,                     /* Key */
      2 iquant real fixed decimal (5,0) aligned /* 9-bit */;

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

SECTION 5

BUILT-IN AND INSTALLATION-DEFINED FUNCTIONS

BUILT-IN FUNCTIONS

The following built-in functions are available in MRDS. Each of the functions is described in detail following the list.

abs	mod
after	reverse
before	round
ceil	search
concat	substr
floor	verify
index	

Built-in functions within a selection expression must be enclosed with square brackets []. For example:

```
... [substr (E.name 1 1)] ...
```

The examples below use the data base described by the model:

```
domain:
     x float bin(27),
     y float dec(27),
     c bit(3);

relation:
     r (x* y c);
```

that contains the following tuple:

```
<5.25 5.25 "101"b>
```

In addition, the following PL/I structure is used as the program fragments:

```
dcl 01 r
     02 x float bin(27),
     02 y float dec(27),
     02 c bit(3);
```

Function: abs

This is an arithmetic scalar function whose reference has the form:

```
abs (X)
```

The result of this function is the absolute value of X, where X must be a numeric data item.

X can only be real and the result value is a float decimal (59).

Function: after

This is a string scalar function whose reference has the form:

```
after (S1 S2)
```

The result is that portion of S1 that occurs after the leftmost occurrence of S2 within S1. If S2 is a null string, the result is S1. If S2 does not occur within S1, the result is a null string. For example:

```
after ("abcde" "bc") = "de"
after ("abcde" "") = "abcde"
after ("abcde" "f") = ""
after ("10101"b "10"b) = "101"b
```

Notes:

When comparing strings, PL/I pads the shorter string on the right. For example:

```
r.c = "101"b;
b1 = (after (r.c, "10"b) = "10"b);
```

results in b1 having a value of "1"b.

MRDS, however, never pads. That is,

```
mrc retrieve 1 1 "-range (r rel)
                  -select r.c
                  -where [after (r.c, ""10""b)] = ""10""b"
```

does not retrieve any tuples.

Function: before

This is a string scalar function whose reference has the form:

```
before (S1 S2)
```

The result is that portion of S1 that occurs before the leftmost occurrence of S2 within S1. If S2 is a null string, the result is a null string. If S2 does not lie within S1, then the result is S1. For example:

```
before ("abcde" "bc") = "a"
before ("abcde" "") = ""
before ("abcde" "f") = "abcde"
before ("10101"b "10"b) = ""b
```

The before function has an anomaly similar to the one described under "Notes" for the after function.

Function: ceil

This is an arithmetic scalar function whose reference has the form:

```
ceil (X)
```

where X must be real. The result is the smallest integer (I) such that:

```
I >= X
```

For example:

```
ceil (20.5) = 21
ceil (-14.6) = -14
ceil (12) = 12
```

Function: concat

This is a string scalar function whose reference has the form:

```
concat (S1 S2)
```

The result is the concatenation of S1 and S2. For example:

```
concat ("abc" "de") = "abcde"
concat ("101"b "01"b) = "10101"b
```

Function: floor

This is an arithmetic scalar function whose reference has the form:

```
floor (X)
```

where X is real. The result is the largest integer (I) such that:

```
I <= X
```

For example:

```
floor (20.5) = 20
floor (-14.6) = -15
floor (12) = 12
```

Function: index

This is a string scalar function whose reference has the form:

```
index (S1 S2)
```

The result is an integer that is the position of the beginning of the leftmost occurrence of S2 within S1. If S2 is not in S1 then the result is 0. If S2 is a null string, the result is 0. For example:

```
index ("abcde" "bc") = 2
index ("abcde" "f") = 0
index ("abcde" "") = 0
```

Function: mod

This is an arithmetic scalar function whose reference has the form:

```
mod (X Y)
```

where X and Y are real. The result is X modulus Y, such that:

```
if Y ^= 0 then mod (X Y) = X - Y * floor (X / Y)
if Y = 0 then mod (X Y) = X
```

For example:

```
mod (42 5) = 2
mod (129.2867 25) = 4.2867
mod (10 0) = 10
```

Function: reverse

This is a string scalar function whose reference has the form:

```
reverse (S)
```

The result is a string which is the reverse of the value of S. For example:

```
reverse ("abcde") = "edcba"
reverse ("a") = "a"
reverse ("") = ""
reverse ("10110"b) = "01101"b
```

Function: round

This is an arithmetic scalar function whose reference has the form:

```
round (X Q)
```

The result is a rounding of the value of X. When a value is rounded to n digits, the digits after the nth digit are dropped and the nth digit is increased by 1 if the (n+1)th digit is 5 or greater. If X is float, then Q must be positive and the mantissa is rounded to Q digits. If X is fixed, it is rounded to a value that has Q fractional digits. For complex values, the function is defined by:

```
round (X + Yi Q) = round (X Q) + round (Y Q)i
```

For example:

```
round (183.629e6 4) = 183.6e6
round (183.629 2) = 183.63
round (183.629 -1) = 180
round (21.56 + 6.21i 0) = 22 + 6i
```

Notes:

If used in PL/I, a binary variable is rounded based on a bit and a decimal variable is rounded based on a digit. For example:

```
r.x = 5.25
round (r.x, 2) = 6.0

r.y = 5.25
round (r.y, 2) = 5.3
```

MRDS always converts the value to be rounded to float decimal before rounding so that:

```
mrc retrieve 1 1 "-range (r rel)
                  -select r.x
                  -where [round (r.x, 2)] = 6"
```

does not retrieve any tuples.

Function: search

This is a character string scalar function whose reference has the form:

```
search (C1 C2)
```

The result is an integer value that is the position in C1 of the leftmost occurrence of any character contained in C2. If C1 does not contain any character in C2, the result is 0. For example:

```
search ("abcde" "b") = 2
search ("abcde" "") = 0
search ("abcde" "f") = 0
search ("abcde" "be") = 2
```

Function: substr

This is a string scalar function whose reference has the form:

```
substr (S I J)
```

-or-

```
substr (S I)
```

The result is that portion of S that begins with the Ith character and has length J (if J is present), or is that portion of S that begins with the Ith character and continues to the end of S (if J is not present). For example:

```
substr ("abcde" 3 2) = "cd"
substr ("abcde" 3 0) = ""
substr ("abcde" 3) = "cde"
substr ("10101"b 3) = "101"b
```

Function: verify

This is a character string scalar function whose reference has the form:

```
verify (C1 C2)
```

The result is an integer value that is the position of the first character of C1 that does not occur in C2. When C1 contains only characters that are in C2, the result is 0. For example:

```
verify ("xyz" "abc") = 1
verify ("xyz" "xyz") = 0
verify ("abcde" "cba") = 4
```

WRITING NONSTANDARD FUNCTIONS

Nonstandard (or installation-defined) functions may be written in PL/I, COBOL, or FORTRAN. It is assumed that these functions are written by experienced programmers. (Refer to the "dsl_$declare" subroutine entry in Section 4 for a description of how to declare user-defined functions.)

Scalar functions are passed a complete standard Multics argument list containing argument pointers and descriptor pointers for both the input arguments and the return argument. The call is equivalent to:

```
return_val = fn_name$fn_name (in_arg1, ... , in_argn);
```

Two restrictions on arguments to nonstandard functions are:

1. No * extents are permitted.
2. Data types are restricted to those data types permitted in a MRDS data base. The use of pointers, entries, labels, structures, offsets, and arrays is not allowed.

Example:

```
user_substr: proc (param) returns (char(30));
dcl param char(30);
     return (substr (param, 1, 6));
end user_substr;
```

SECTION 6

SUBSYSTEM WRITERS' GUIDE

The MRDS Subsystem Writers' Guide is a reference of interest to writers of sophisticated subsystems. It documents user-accessible modules that allow the user to bypass standard MRDS facilities. The interfaces are a level deeper into the system than those required by the majority of users.

The MRDS Subsystem Writers' Guide provides the advanced Multics user a selection of some of the internal interfaces used to construct the standard MRDS user interface.

An example of a specialized subsystem that requires reference to the MRDS Subsystem Writers' Guide for its construction is a subsystem intended to provide end-user access to a MRDS data base.

The subroutines contained in this section are: mmi_ and msmi_.

Name: mmi_

This subroutine primarily provides a means of retrieving information about a data base model (Mrds_Model_Interface_). There is also an entry to create a data base in the same manner as the create_mrds_db command. This interface replaces dmd_ which is obsolete (see Section 10). See the msmi_ subroutine interface for submodel information.

Entry: mmi_$close_model

This entry closes a given opening of the data base model.

Usage

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

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

where:

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

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

Entry: mmi_$create_db

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

Usage

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

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

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

Notes

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

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

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

Entry: mmi_$get_authorization

This entry returns the user class of the caller for a given data base.

Usage

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

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

where:

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

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

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

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

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

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

   error_table_$badcall
   if the area_ptr was null.

mrds_error_$no_data base
if the given path does not refer to a MRDS data base.

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

error_table_$unimplemented_version
if the given structure version is unknown.

mrds_error_$version_not_supported
if the data base path does not refer to a version 4 MRDS data base.

Notes

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

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

where:

1. version
   is the version number of this structure, which should be set by calling mrds_authorization_structure_version.

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

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

4. mbz
   is reserved for future use.

Currently, the only available structure version is 1.

The following variables

mrds_authorization_ptr
mrds_authorization_structure_version

are also declared by the mrds_authorization include file.

A DBA is currently defined as the holder of "sma" access on the data base directory.

Entry: mmi_$get_model_attributes

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

Usage

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

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

where:

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

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

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

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

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

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

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

   error_table_$badcall
   if the area_ptr was null.

   error_table_$unimplemented_version
   if the structure version given was unknown.

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

   mrds_error_$no_model_rel
   if the relation name given is not in the model definition.

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

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

Notes

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

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

where:

1. version
   is the version number of this structure, which should be set by calling mrds_db_model_rel_attrs_structure_version.

2. attribute_count
   is the number of attributes in this relation.

3. mbz1
   is reserved for future use.

4. name
   is the name of this attribute.

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

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

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

8. mbz2
   is reserved for future use.

Currently the only structure version available is 1.

The variables mrds_db_model_attrs_ptr, mrds_db_model_rel_attrs_count_init, and mrds_db_model_rel_attrs_structure_version are also declared in the mrds_db_model_rel_attrs include file.

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

Entry: mmi_$get_model_info

This entry returns information about the data base model creation.

Usage

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

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

where:

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

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

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

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

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

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

   error_table_$badcall
   if the area_ptr was null.

   error_table_$unimplemented_version
   if the supplied structure version is unknown.

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

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

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

Notes

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

```
dcl 1 mrds_db_model_info aligned,
    2 version fixed bin,
    2 model_version fixed bin,
    2 db_type fixed bin,
    2 dmfile_ attributes,
      3 protected bit(1) unal,
      3 rollback bit(1) unal,
      3 concurrency bit(1) unal,
      3 mbz bit(33) unal,
    2 creator_id char(32),
    2 creation_time fixed bin(71);
```

where:

1. version
   is the version number of this structure, which should be set by calling mrds_db_model_info_structure_version.

2. model_version
   is the data base version. The latest version is 4.

3. db_type
   indicates the type of data base. A value equal to mrds_db_model_info_vfile_type indicates a vfile type data base while a value equal to mrds_db_model_info_dmfile_type indicates a dmfile type data base. The variables mrds_db_model_info_vfile_type and mrds_db_model_info_dmfile_type are declared in the mrds_db_model_info include file.

4. protected
   a value of "1"b indicates that a transaction must be in progress to reference the data in the data base; a value of "0"b indicates that transactions are not needed. This field will always have a value of "0"b if the data base is a vfile type data base.

5. rollback
   a value of "1"b indicates that a before journal will be used to journalize transaction activity; a value of "0"b indicates that a before journal will not be used. This field will always have a value of "0"b if the value of the protected element is also "0"b.

6. concurrency
   a value of "1"b indicates that locking will be done at the control interval level; a value of "0"b indicates that locking will not be done at the control interval level. This field will always have a value of "0"b if the value of the protected element is also "0"b.

7. mbz
   these bits must be zero (for future use).

8. creator_id
   is in the form Person_id.Project_id.tag as returned from get_group_id_ for the creator of the data base.

9. creation_time
is the time the data base was created in a form acceptable to the date_time_ subroutine.

The latest version of the structure is version 2. Programs using the version 1 structure will continue to execute correctly.

The variables mrds_db_model_info_ptr and mrds_model_info_structure_version are also declared in the mrds_db_model_info include file.

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

Entry: mmi_$get_model_relations

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

Usage

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

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

where:

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

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

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

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

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

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

   error_table_$badcall
   if the area_ptr was null.

   error_table_$unimplemented_version
   if the given structure version is unknown.

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

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

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

Notes

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

```
dcl 1 mrds_db_model_relations aligned
      based (mrds_db_model_relations_ptr),
    2 version,
    2 relation_count fixed bin,
    2 mbz1 bit(36) unal,
    2 relation (0
      refer (mrds_db_model_relations.relation_count)),
```

```
          3 name char(32),
          3 mbz2 bit(36) unal;
```

where:

1. version
   is the version number of this structure, which should be set by calling mrds_db_model_relation_structure_version.

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

3. mbz1
   is reserved for future use.

4. name
   is the name of this relation.

5. mbz2
   is reserved for future use.

Currently, the only structure version available is 1.

The variables

```
mrds_db_model_relation_ptr
mrds_db_model_relation_count_init
mrds_db_model_relation_structure_version
```

are also declared in the mrds_db_model_relations include file.

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

Entry: mmi_$get_secured_state

This entry returns the secured state of the given data base.

Usage

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

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

where:

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

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

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

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

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

   error_table_$area_too_small
   if the supplied area could not hold the data base state information.

   error_table_$badcall
   if the area_ptr was null.

   error_table_$insufficient_access
   if the user has no access to both the data base directory and the db_model segment.

   error_table_$unimplemented_version
   if the supplied structure version is unknown.

   mrds_error_$no_data base
   if the given path does not refer to a MRDS data base.

   mrds_error_$no_model_access
   if the user does not have "r" access to the data base db_model segment.

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

mrds_error_$version_not_supported
if the path given is to a data base whose version is less than 4.

Notes

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

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

where:

1. version
   is the version number of this structure, which should be set by calling database_state_structure_version.

2. unsecured
   is "1"b if the data base is not currently secured.

3. secured
   is "1"b if the data base is currently secured.

4. mbz
   is reserved for future use.

Currently, the only structure version available is one.

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

Entry: mmi_$open_model

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

Usage

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

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

where:

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

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

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

   error_table_$insufficient_access
   if the data base has been secured and the user is not a DBA.

   mrds_error_$no_database
   if no data base exists at the given pathname.

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

   mrds_error_$open_name_already_known
   if the opening_name supplied was not unique, within PL/I comparison rules, compared to other opening names already used in the user's process.

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

Notes

The opening_name may be any number of ASCII characters. Current capability is for more than 1000 opening_names of reasonable length for version 4 models, but only 64 for models of version 3. Opening_names must be unique to PL/I comparison rules within the user's process. (The entry unique_chars_, described in MPM Subroutines, can be used to generate unique names.)

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

Entry: mmi_$quiesce_db

This entry allows the DBA to quiesce a given data base for such purposes as data base backup or other exclusive activities that require a consistent and non-active data base. The data base can be returned to a non-quiescent state by use of the mmi_$unquiesce_db entrypoint.

Usage

```
declare mmi_$quiesce_db entry (char(*), fixed bin(17), fixed bin(35));

call mmi_$quiesce_db (database_path, wait_time, error_code);
```

where:

1. database_path (Input) (char(*))
   is the relative or absolute pathname of the data base to be quiesced. Version 4 data bases need not have the db suffix supplied.

2. wait_time (Input) (fixed bin(17))
   sets the amount of time that an attempt to quiesce waits for conflicting data base users to depart before failing (see "Notes").

3. error_code (Output) (fixed bin(35))
   is the standard status code.

Note

Time specified for wait_time is in seconds. A long wait is needed if the data base is open by many users; otherwise, a short wait_time will suffice. For a simple go/nogo test, give a wait_time of 1 second.

Entry: mmi_$unquiesce_db

This entry returns a data base that is in a quiescent state (by either mmi_$quiesce_db or the quiesce_mrds_db command) to a non-quiescent state.

Usage

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

call mmi_$unquiesce_db (database_path, error_code);
```

where:

1. database_path (Input) (char(*))
   is the relative or absolute pathname of the data base to be freed. Version 4 data bases need not have the db suffix supplied.

2. error_code (Output) (fixed bin(35))
is the standard status code.

Name: msmi_

This is a subroutine interface to the MRDS submodel data structure (mrds_submodel_interface). The submodel data structure is created by the create_mrds_dsm command and may be displayed by the display_mrds_dsm command. This interface replaces the obsolete dsmd_ interface (see Section 10).

Entry: msmi_$close_submodel

This entry disassociates an opening name and a submodel to prevent further access to that submodel through that opening_name.

Usage

```
declare msmi_$close_submodel entry (char(*), fixed bin(35));

call msmi_$close_submodel (opening_name, code);
```

where:

1. opening_name (Input) (char(*))
   is the name identifying the submodel opening.

2. code (Output) (fixed bin(35))
   is a standard error code.

Notes

The submodel-opening_name association must already have been made by a successful call to msmi_$open_submodel. If the opening_name is not known, the error code mrds_error_$open_name_not_known is returned.

Entry: msmi_$get_attribute_data

This entry returns the attribute information for the given relation.

Usage

```
declare msmi_$get_attribute_data entry (char(*), char(*), ptr, fixed bin,
     ptr, fixed bin(35));

call msmi_$get_attribute_data (opening_name, rel_name, area_ptr,
     str version, attribute data ptr, code);
```

where:

1. opening_name (Input) (char(*))
   is the name identifying the submodel opening.

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

3. area_ptr (Input) (ptr)
   is a pointer to a freeing area where the mrds_dsm_attribute_data structure will be allocated.

4. str_version (Input) (fixed bin)
   is the version of the mrds_dsm_attribute_data structure that is to be allocated.

5. attribute_data_ptr (Output) (ptr)
   is a pointer to the allocated structure.

6. code (Output) (fixed bin(35))
   is a standard error code.

Notes

The submodel-opening_name association must already have been made by a successful call to msmi_$open_submodel. If the opening_name is not known, the error code mrds_error_$open_name_not_known is returned.

If the area pointed to by the area_ptr parameter is too small for the mrds_dsm_attribute_data structure to be allocated in it, the error code error_table_$area_too_small is returned. If the area_ptr parameter is null, the error code error_table_$badcall is returned. If the area is not a freeing area, the error code mrds_error_$not_freeing_area is returned.

The following is version 1 (currently the only version) of the mrds_dsm_attribute_data structure (see Appendix F for the include file mrds_dsm_attribute_data.incl.pl1). If the str_version parameter refers to a version of the mrds_dsm_attribute_data structure that is not supported or does not exist, the error code error_table_$unimplemented_version is returned.

```
dcl 1 mrds_dsm_attribute_data based
      (mrds_dsm_attribute_data_ptr) aligned,
    2 version fixed bin,
    2 number_of_attributes fixed bin,
    2 attributes (mrds_dsm_attribute_data_num_atts refer
      (mrds_dsm_attribute_data.number_of_attributes)),
      3 submodel_attribute_name char(64),
      3 model_attribute_name char(32),
      3 read_access bit(1) unal,
      3 modify_access bit(1) unal,
      3 null_access bit(1) unal,
      3 mbz1 bit(33) unal;
```

where:

1. version
   is the version of the structure, which should be set by calling mrds_dsm_attribute_data_structure_version.

2. number_of_attributes
   is the number of attributes in the submodel relation view.

3. submodel_attribute_name
   is the name of the attribute in the submodel.

4. model_attribute_name
   is the name of the attribute in the model.

5. read_access
   is set to "1"b if the submodel has read access set for the attribute.

6. modify_access
   is set to "1"b if the submodel has modify access set for the attribute.

7. null_access
   is set to "1"b if the submodel has null access set for the attribute.

8. mbz1
   is set to "0"b.

The variables

```
mrds_dsm_attribute_data_ptr
mrds_dsm_attribute_num_atts
mrds_dsm_attribute_data_structure_version
```

are also declared in the mrds_dsm_attribute_data include file.

If the submodel refers to a secure data base and the user calling msmi_$get_attribute_data is not a data base administrator for the data base, then the value of model_attribute_name will be null.

If null_access has a value of "1"b then both read_access and modify_access will have values of "0"b.

Entry: msmi_$get_relation_data

This entry returns information about each relation in the submodel.

Usage

```
declare msmi_$get_relation_data entry (char(*), ptr, fixed bin, ptr,
     fixed bin(35));

call msmi_$get_relation_data entry (opening_name, area_ptr, str_version,
     relation_data_ptr, code);
```

where:

1. opening_name (Input) (char(*))
   is the name identifying the submodel opening.

2. area_ptr (Input) (ptr)
   is a pointer to a freeing area where the mrds_dsm_relation_data structure will be allocated.

3. str_version (Input) (fixed bin)
   is the version of the mrds_dsm_relation_data structure that is to be allocated.

4. relation_data_ptr (Output) (ptr)
   is a pointer to the allocated structure.

5. code (Output) (fixed bin(35))
   is a standard error code.

Notes

The submodel-opening_name association must already have been made by a successful call to msmi_$open_submodel. If the opening_name is not known, the error code mrds_error_$open_name_not_known is returned.

If the area pointed to by the area_ptr parameter is too small for the mrds_dsm_relation_data structure to be allocated in it, the error code error_table_$area_too_small is returned. If the area_ptr parameter is null, the error code error_table_$badcall is returned. If the area is not a freeing area, the error code mrds_error_$not_freeing_area is returned.

The following is version 1 (currently the only version) of the mrds_dsm_relation_data structure (see Appendix F for the include file mrds_dsm_relation_data.incl.pl1). If the str_version parameter refers to a version of the mrds_dsm_relation_data structure that is not supported or does not exist, the error code error_table_$unimplemented_version is returned.

```
dcl 1 mrds_dsm_relation_data based
      (mrds_dsm_relation_data_ptr) aligned,
    2 version fixed bin,
    2 number_of_relations fixed bin,
    2 relations (mrds_dsm_relation_data_num_rels refer
      (mrds_dsm_relation_data.number_of_relations)),
      3 submodel_relation_name char(64),
      3 model_relation_name char(32),
      3 append_access bit(1) unal,
      3 delete_access bit(1) unal,
      3 null_access bit(1) unal,
      3 mbz1 bit(36) unal;
```

where:

1. version
   is the version of the structure.

2. number_of_relations
   is the number of relations in the submodel.

3. submodel_relation_name
   is the relation name defined in the submodel.

4. model_relation_name
   is the corresponding name of the relation as defined in the model.

5. append_access
   is set to "1"b if the submodel has append access set for the relation.

6. delete_access
   is set to "1"b if the submodel has delete access set for the relation.

7. null_access
   is set to "1"b if the submodel has null access set for the relation.

8. mbz1
   is set to "0"b.

The variables

```
mrds_dsm_relation_data_ptr
mrds_dsm_relation_data_num_rels
mrds_dsm_relation_data_structure_version
```

are also included in the mrds_dsm_relation_data include file.

If the submodel refers to a secure data base and the user calling msmi_$get_relation_data is not a data base administrator for the data base, then the value of model_relation_name will be null.

If null_access has a value of "1"b then both append_access and delete_access will have values of "0"b.

Entry: msmi_$get_submodel_info

This entry returns general information about the submodel.

Usage

```
declare msmi_$get_submodel_info entry (char(*), ptr, fixed bin, ptr,
     fixed bin(35));

call msmi_$get_submodel_info (opening_name, area_ptr, str_version,
     submodel_info_ptr, code);
```

where:

1. opening_name (Input) (char(*))
   is the name identifying the submodel opening.

2. area_ptr (Input) (ptr)
   is a pointer to a freeing area where the mrds_dsm_submodel_info structure will be allocated.

3. str_version (Input) (fixed bin)
   is the version of the mrds_dsm_submodel_info structure that is to be allocated.

4. submodel_info_ptr (Output) (ptr)
   is a pointer to the allocated structure.

5. code (Output) (fixed bin(35))
   is a standard error code.

Notes

The submodel-opening_name association must already have been made by a successful call to msmi_$open_submodel. If the opening_name is not known, the error code mrds_error_$open_name_not_known is returned.

If the area pointed to by the area_ptr parameter is too small for the mrds_dsm_submodel_info structure to be allocated in it, the error code error_table_$area_too_small is returned. If the area_ptr parameter is null, the error code error_table_$badcall is returned. If the area is not a freeing area, the error code mrds_error_$not_freeing_area will be returned.

The following is version 1 (currently the only version) of the mrds_dsm_submodel_info structure (see Appendix F for the include file mrds_dsm_submodel_info.incl.pl1). If the str_version parameter refers to a version of the mrds_dsm_submodel_info structure that is not supported or does not exist the error code error_table_$unimplemented_version will be returned.

```
dcl 1 mrds_dsm_submodel_info based
      (mrds_dsm_submodel_info_ptr) aligned,
    2 version fixed bin,
    2 submodel_version fixed bin,
    2 database_path char(168),
    2 submodel_path char(168),
    2 date_time_created fixed bin(71),
    2 creator_id char(32);
```

where:

1. version
   is the version of the structure, which should be set by calling mrds_dsm_submodel_info_structure_version.

2. submodel_version
   is the version of the submodel data structure.

3. datebase_path
   is the absolute path of the data model for which the submodel is defined.

4. submodel_path
   is the absolute path of the submodel.

5. date_time_created
   is the Multics clock value (suitable for input into the date_time_ subroutine) for when the submodel was created.

6. creator_id
   is the ID of the user who created the submodel. It has the form of "Person_id.Project_id.Tag".

The variables

```
mrds_dsm_submodel_info_ptr
mrds_dsm_submodel_info_structure_version
```

are also declared in the mrds_dsm_submodel_info include file.

Entry: msmi_$open_submodel

This entry associates a submodel with an opening_name so that it can be used by other msmi_ entries. The same submodel may be associated with multiple opening names.

Usage

```
declare msmi_$open_submodel (char(*), char(*), fixed bin(35));

call msmi_$open_submodel (opening_name, path, code);
```

where:

1. opening_name (Input) (char(*))
   is the name identifying the submodel opening. This name must be unique within the opening process (as determined by PL/I comparison rules), not only for submodel openings, but for any operation within the MDBM subsystem that takes an opening_name name. Multiple openings of the same submodel must have different opening_name names.

2. path (Input) (char(*))
   is the relative or absolute path (with or without the dsm suffix) of the submodel to be opened.

3. code (Output) (fixed bin(35))
   is a standard error code.

Notes

The opening name can be any length and can be made up of any sequence of ASCII characters. If the opening_name has already been used, the error code

mrds_error_$open_name_already_known is returned. If there is no room to create another opening_name, the error code mrds_error_$too_many_open_names is returned. The exact number of opening_names depends on the length of the names already used, but it is large (> 1000).

SECTION 7

SECURITY

MRDS provides two different levels of data base security: relation level security and attribute level security. The level of security that is enforced depends upon the security state of the data base. The capabilities that a MRDS user has depends not only on the security state of the data base but also upon whether or not the user is a data base administrator (DBA).

DBA

A DBA is a user who has sma ACL on the data base directory. There may be one or several DBAs for a data base; the creator is always a DBA. A DBA is automatically given the necessary Multics ACLs when executing a MRDS command or subroutine.

Secure Data Bases

A data base may be secured (by a DBA) in one of two ways: either by using the secure_mrds_db command with the -set control argument or by creating it in a secure state by using create_mrds_db with the -secure control argument. A data base may be unsecured (by a DBA) by issuing the secure_mrds_db command with the -reset control argument. A secure data base is a data base which has been secured and not subsequently unsecured.

A secure data base cannot be referenced by a non-DBA via either the data model or via an unsecured submodel (a submodel that is not located under the secure.submodels directory). The DBA may reference a secure data base via either the data base's data model or a submodel (either secure or unsecured).

Secure Submodels and the secure.submodels Directory

The secure.submodels directory is a directory located under the data base directory. This directory is used to ensure that a secure data base is referenced through a submodel under the control of a DBA (a secure submodel). A submodel may be placed in this directory either during submodel creation by using the -install control argument or by copying an already created submodel. An unsecured data base may also have some of its submodels in this directory.

A secure data base may be referenced by a non-DBA via a link in some other directory as long as the link's target is a secure submodel. A submodel in some other directory pointed to by a link in the secure.submodels directory is not considered secure.

Required ACLs

The ACLs required on the data model and relation model segments are independent of the level of security in effect. In order to use the data base at all, the user must have "r" ACL on the db_model segment; only DBAs should have "rw" ACL to this segment. When the data base is opened via the data model, the user's view contains all the relations in the data base; if the opening was via a submodel then the view contains just those relations in the submodel. Users will need "r" ACL set on all relation models that correspond to relations in every view to which they have access. No one but DBAs should have "rw" ACL on a relation model segment.

For those submodels on which the DBA controls Multics ACLs (either unsecured submodels contained in a directory created by the DBA or secure submodels contained in the secure.submodels directory of the data base), it is recommended that no one but DBAs have sma ACL on the containing directory and that non-DBA users have "r" ACL only on submodels that they are allowed to use. Only the DBAs should have "rw" ACL on the submodel segments.

The data base consists of three directories: the main data base directory and two inferior directories (resultant_segs.dir and secure.submodels). The general user should have "n" ACL on the data base and the secure.submodels directories while having "s" ACL on the resultant_segs.dir directory. DBAs, by definition, have "sma" ACL set on the data base directory and will automatically be granted "sma" ACL on the inferior directories when executing a MRDS command that requires that ACL.

The ACL required on each relation data segment depends on the operations that will be allowed on that relation and not on the level of security in effect.

1. ACLs of "rw" are required on each relation data segment where the allowed operations are storing a tuple into the relation (append_tuple), deleting a tuple from the relation (delete_tuple), or modifying an attribute value (modify_attr).

2. For a relation where the only permissible operation is to read the attribute values (read_attr), an ACL of "r" is required on the relation data segment.

3. Relation data segments corresponding to relations that have no access permissions should have "n" ACL set.

4. ACLs of "r" should be set on the segments dbcb and rdbi, located in the resultant_segs.dir, for each person alloed to open the data base.

See the command create_mrds_db for a description of the data base makeup in terms of directories and segments.

Scopes

Regardless of the level of security in effect, scopes must be set before any data can be accessed. It is assumed that a scope will be requested only if the indicated operation is to be performed. For this reason, if the requested scope requires more privileges than the user has been assigned (determined from the level of security in effect and the relation's ACL), a data access violation error is generated.

Since an opening in exclusive mode automatically sets scopes, access violation errors may be generated at open time as well.

See the table of access requirements listed under dsl_$set_scope.

LIMITATION: MRDS does not allow a user whose authorization is higher than the access class of a data base to set_scope on that data base.

Relation Level Security

This level of security does not provide any data model security in that there is no restriction on the amount of information about the data base model that the user may obtain. Any user may access the data base via the data model and data access permissions are set at the relation level by using Multics ACLs. This level of security was the only form of security available in MR8.0; it is the only form of security enforceable for an unsecured data base.

All access violations are determined at scope setting time.

- An ACL of "n" will prevent any scope from being set. In effect there is no access to this relation.
- An ACL of "r" will allow scopes with permit_ops of read_attr to be set.
- An ACL of "rw" will allow scopes with permits_ops of read_attr, modify_attr, append_tupld, and delete_tuple to be set.

Attribute Level Security

Attribute Level Security, which is enforced only for a secure data base, provides both data value security and data model security.

DATA MODEL SECURITY

In a secure data base, users are granted access to a subset of the data base. In order to prevent these users from obtaining more information about the data base than their view allows, the following commands and subroutines, which deal exclusively with the data model, are restricted to DBAs if the data base is in a secured state:

```
create_mrds_dsm
display_mrds_dm
dmd_
mmi_
```

For an unsecured data base, dsl_$open and commands that may operate on either the data model or a data submodel are restricted to secure submodels when used by a non-DBA on a secure data base. These commands are:

```
create_mrds_dm_include
create_mrds_dm_table
display_mrds_db_status
```

Many commands and subroutines either display or return the relation and attribute data model names associated with the submodel relation and attribute names. A non-DBA user invoking one of these commands/subroutines will have spaces (blanks) displayed/returned in place of the model relation and attribute names if the associated data base is in a secured state.

```
display_mrds_db_access
display_mrds_dsm
dsmd_
msmi_
```

To use the data value security features of attribute level security the data base must not only be secured but there must also be at least one secured submodel containing specifications for the data access permissions on both the relations in the view defined by the submodel and the attributes in those relations (see the create_mrds_dsm command).

The access permissions that may be set in the submodel correspond to the scopes that may be requested, i.e:

relation access:
append_tuple
delete_tuple
null

attributes access:
read_attr
modify_attr
null

The only restriction on the attribute access permissions is that null access cannot be specified with any other access.

There are several restrictions on which access can be set on the relations.

1. null relation access permission cannot be specified with any other access permissions.

2. append_tuple and delete_tuple can only be set if the submodel relation is a full view of its corresponding model relation. A full view implies that the submodel relation contains all the attributes in the model relation.

3. append_tuple can only be set on a relation if read_attr access is also set on all of the primary key attributes in that relation. If this restriction were not applied then it would be possible to store tuples until the duplicate key error was generated, at which point the values of the primary key attributes would be known.

Because the access specified in the submodel is independent of the Multics ACLs on the relation's data segments, Multics will enforce the ACLs on those segments. It is the responsibility of a DBA to make sure that the ACLs on the relation data segments allow the submodel user to perform the operations set in the submodel. Specifically for those users with "r" ACL on the submodel:

1. For every relation in the submodel with either append_tuple or delete_tuple access or with at least one attribute with modify_attr access, set "rw" ACL on the relation data segment.

2. For every relation in the submodel with null access on the relation and with only read_attr access on the attributes, set "r" ACL.

Relation model segments should have "r" ACL set, for all relations appearing in the submodel view.

Unlike Relation Level Security, not all access violations can be detected at set_scope time. This is because scopes are set at the relation level, but the access specifications are at the attribute level.

*

Example

The following submodels are all based on the employee_db data model (described below). Each submodel is used by one project, which has the same name as the submodel (i.e. *.submodel_name.*). For each submodel, the submodel source, the display produced by display_mrds_dsm, and the required Multics ACLs on all data base entries for the project are listed. A table listing the effective access for each relation and attribute for both a secured and unsecured data base is also given. Note that an unsecured data base may be opened via the submodel or via the data model.

The cmdb source of the employee_db data base is presented first, followed by a display of the ACLs on all the entries associated with the data base for the payroll, credit_union, and DBA projects. That is followed by the displays for each submodel/project.

The Employee_db Source

```
domain:
     birth_date      fixed bin(71),
     city            char(32),
     credit_union    fixed decimal(6, 2),
     end_date        fixed bin(71),
     end_sal         fixed decimal(6, 2),
     federal         fixed decimal(6,2),
     first_name      char(32) varying,
     id              fixed bin(17),
     job_title       char(64),
     last_name       char(64) varying,
     pay             fixed decimal(6, 2),
     pension         fixed decimal(6, 2),
     sex             char(1),
     ssn             char(11),
     start_date      fixed bin(71);
     start_sal       fixed dec(6, 2),
     state           char(32);
     state_with      fixed decimal(6, 2),
     street          char(32);
     zip             char(9),

relation:
     address         (id* street city state zip),
     payroll         (id* pay federal state_with pension credit_union ssn),
     personal        (id* first_name last_name sex birth_date spouse_name);

index:
     address         (zip),
     personal        (last_name birth_date);
```

Summary of all Multics ACLs for all data base entries:

```
employee_db.db
     sma        *.DBA.*

db_model
     rw         *.DBA.*
     r          *.payroll.*
     r          *.credit_union.*

db.control
     rw         *.DBA.*
     rw         *.payroll.*
     rw         *.credit_union.*
```

```
address.m
     rw         *.DBA.*
     r          *.payroll.*
     r          *.credit_union.*

address
     rw         *.DBA.*
     r          *.payroll.*
     r          *.credit_union.*

payroll.m
     rw         *.DBA.*
     r          *.payroll.*
     r          *.credit_union.*

payroll
     rw         *.DBA.*
     rw         *.payroll.*
     rw         *.credit_union.*

personal.m
     rw         *.DBA.*
     r          *.payroll.*
     r          *.credit_union.*

personal
     rw         *.DBA.*
     r          *.payroll.*
     r          *.credit_union.*

secure.submodels
     sma        *.DBA.*

secure.submodels>payroll.dsm
     r          *.payroll.*

secure.submodels>credit_union.dsm
     r          *.credit_union.*

resultant_segs.dir
     sma        *.DBA.*

resultant_segs.dir>dbcb
     rw         *.DBA.*

resultant_segs.dir>rdbi
     rw         *.DBA.*
```

The Payroll Submodel

The payroll submodel allows access to the address, payroll, and part of the personal relations. The ACLs on these relations only allow tuples to be appended to or deleted from the payroll relation and only in the payroll relation may attribute values be modifed. If the data base is secured, then the credit_union and pension attributes of the payroll relation may not be modified. If the data base is not secured, it is not possible for MRDS to prevent the modification of the credit_union and pension values or to prevent the reading of the sex, birth_date and spouse_name attributes in the personal relation.

The source:

```
relation:           address   (id street city state zip);

relation access:    payroll   (append_tuple, delete_tuple)
                                with attribute access
                                  (read_attr, modify_attr);
```

```
relation:           payroll   (id pay federal state_with
                                 credit_union pension ssn);
attribute access:   pension in payroll (read_attr),
                    credit_union in payroll (read_attr);

relation:           personal  (id last_name first_name);
```

```
display_mrds_dsm >examples>employee_db.db>secure.submodels>payroll.dsm
```

```
          address             n
            id                r
            street            r
            city              r
            state             r
            zip               r

          payroll             ad
            id                rm
            pay               rm
            federal           rm
            state_with        rm
            credit_union      r
            pension           r
            ssn               rm

          personal            n
            id                rm
            last_name         r
            first_name        r
```

Required ACLs for *.payroll.*

```
employee_db.db      n
db_model            r
db.control          rw
address.m           r
address             r
payroll.m           r
payroll             rw
personal.m          r
personal            r
secure.submodels    n

secure.submodels>payroll.dsm            r
secure.submodels>credit_union.dsm       n
resultant_segs.dir>rdbi                 n
resultant_segs.dir>dbcb                 n
```

Effective access for the *.payroll.*

```
relation      unsecured    secured
 attribute  model  submodel  submodel

address       n     n         n
 id           r     r         r
 street       r     r         r
 city         r     r         r
 state        r     r         r
 zip          r     r         r

payroll       ad    ad        ad
 id           rm    rm        rm
```

```
 pay           rm     rm          rm
 federal       rm     rm          rm
 state_with    rm     rm          rm
 credit_union  rm     rm          r
 pension       rm     rm          r
 ssn           rm     rm          rm

personal       n      n           n
 id            r      r           r
 last_name     r      r           r
 first_name    r      r           r
 sex           r      not_visable
 birth_date    r      not_visable
 spouse_name   r      not_visable
```

The Credit Union Submodel

The credit union submodel allows the credit union office to read the address relation and parts of the payroll and personal relations. In addition, the credit_union attribute in the payroll relation may be modified. Note that, if the data base is not secured, credit union personnel may open the data base via the data model, read all the attributes in the personal relation, and read and modify all the attributes in the payroll relation as well as add and delete tuples.

The source:

```
relation:            address   (id street city state zip);

relation:            payroll   (id ssn credit_union);
attribute access:    credit_union in payroll (read_attr, modify_attr);

relation:            personal  (id last_name first_name);
```

```
display_mrds_dsm >example>employee_db.db>secure.submodels>credit_union.dsm

address              n
  id                 r
  street             r
  city               r
  state              r
  zip                r

payroll              n
  id                 r
  ssn                r
  credit_union       rm

personal             n
  id                 r
  last_name          r
  first_name         r
```

Required ACLs for *.credit_union.*

```
employee_db.db       n
db_model             r
```

```
db.control              rw
address.m               r
address                 r
payroll.m               r
payroll                 rw
personal.m              r
personal                r
secure.submodels        n

secure.submodels>payroll.dsm            n
secure.submodels>credit_union.dsm       r
resultant_segs.dir>rdbi                 n
resultant_segs.dir>dbcb                 n
```

Effective access for *.credit_union.*

relation attribute	unsecured model	secured submodel	secured submodel
address	n	n	n
id	r	r	r
street	r	r	r
city	r	r	r
state	r	r	r
zip	r	r	r
payroll	ad	n	n
id	rm	rm	r
pay	rm	not_visable	
federal	rm	not_visable	
state_with	rm	not_visable	
credit_union	rm	rm	rm
pension	rm	not_visable	
ssn	rm	rm	r
personal	n	n	n
id	r	r	r
last_name	r	r	r
first_name	r	r	r
sex	r	not_visable	
birth_date	r	not_visable	
spouse_name	r	not_visable	

TABLE OF EFFECTS OF DATA BASE SECURITY

DBA Only

adjust_mrds_db
create_mrds_dsm -install option
secure_mrds_db
quiesce_mrds_db

Secured DB - DBA Only

create_mrds_dsm
display_mrds_dm
dmd_
mmi_

Secured DB - Restricted to Secured Submodels for Non-DBA

create_mrds_dm_include
create_mrds_dm_table
display_mrds_db_status
dsl_$open
mrds_call open

Secured DB - Access Violation Detection/Display Change

display_mrds_db_access
display_mrds_dsm
dsl_$open
dsl_$set_scope
msmi_
mrds_call open
mrds_call set_scope

Unaffected

display_mrds_db_version
display_mrds_open_dbs
display_mrds_scope_settings
display_mrds_temp_dir
dsl_ (other than open or set_scope)
dsmd_
mrds_call (other than open or set_scope)
set_mrds_temp_dir
update_mrds_db_version

New Options

create_mrds_db -secure option
create_mrds_dsm -install option

```
display_mrds_db_status
display_mrds_scope_settings
dsl_$get_scope
mrds_call get_scope
```

SECTION 8

DATA BASE BACKUP

This section describes the procedure for providing a backup copy of a data base and its data. This is an administrative facility for use by the DBA and is not intended for use by the general data base user community.

CHECKPOINT

To make a backup (checkpoint) copy of a data base, perform the following steps:

1. Do a quiesce_mrds_db on the current data base to ensure a consistent copy and data base integrity.
2. Do a copy_dir of the data base to another part of the storage hierarchy to obtain the checkpoint data base.
3. Do a quiesce_mrds_db using the -free option on the original data base.

ROLLBACK

To return the current data base to its previous checkpointed state, perform the following steps:

1. Do a quiesce_mrds_db of the current data base.
2. Do a delete_dir of the current data base.
3. Do a copy_dir of the checkpoint data base into the current location of the current data base.
4. Do a quiesce_mrds_db using the -free option on the new current data base.

Notes

Doing a display_mrds_db_status of the checkpoint data base shows it as quiesced. (It should be kept that way to ensure integrity.)

The copy can have itself freed from quiescing if it is to be used for program development rather than as a backup copy.

If a display_mrds_db_status of the current data base shows many users, a long wait time should be used for quiesce_mrds_db. With few users or none, a small wait time will suffice.

The quiescing process is the only process that can open the checkpointed, quiesced data base.

If the data base was quiesced by a dead process, both quiesce_mrds_db using the -free option and adjust_mrds_db using the -trouble_switch off option must be done. However, there is danger of an inconsistent data base, depending on why the quiescing process died.

SECTION 9

DATA BASE DEVELOPMENT TOOLS

This section describes the mrds_call command and its related functions, which may be used in the development phase of a data base application. They are normally not used in a production environment.

The following is a summary of mrds_call functions.

close, c
: closes the currently open data bases.

declare, dcl
: makes a user-defined function known to MRDS for possible use as a -where clause.

define_temp_rel, dtr
: redefines or creates a new temporary relation which can be accessed by the current process.

delete, dl
: specifies that the selected data is to be deleted from the data base.

dl_scope, ds
: deletes all or a portion of the current scope of access.

dl_scope_all, dsa
: deletes all of the current scope of access.

get_population, gp
: returns the number of tuples currently stored in a permanent or temporary relation.

get_scope, gs
: displays the scope currently set on a relation in a given opening of the data base.

list_dbs, ld
: displays the indexes, pathnames, and opening modes of all data bases currently open by the user via mrds_call only.

modify, m
: specifies that the selected portion of the data base is to be modified.

open, o
: opens the specified data bases or data submodels for processing.

retrieve, r
: displays the selected data specified by the selection expression.

set_modes, sm
: allows setting of the amount of error information returned and/or the displaying of opening information.

set_scope, ss
: defines the current scope of access for a relation within a shared data base.

set_scope_all, ssa
: defines the current scope of access for all relations within a shared data base.

store, s
: adds a new tuple (row) to the selected relation.

Name: mrds_call, mrc

This command provides a command-level interface to the Data Sublanguage (DSL). It is not intended to be a true end-user query language, but rather is designed to be used as an experimentation vehicle for data base administrators and applications programmers during the development of a data base and its associated programs. The mrds_call command is also useful as an instructional tool when introducing new users to MRDS. Refer to the LINUS Manual for an actual end-user facility which accesses MRDS data bases.

Usage

mrds_call function-name {args}

where:

1. function-name
   is one of the following functions:

close	list_dbs
declare	modify
define_temp_rel	open
delete	retrieve
dl_scope	set_modes
dl_scope_all	set_scope
get_scope	set_scope_all
get_population	store

2. args
   are arguments that depend on the particular function to be performed. Specific arguments are described below under the functions with which they can be used. Each argument is limited to 256 characters in length.

Usage of this command is explained below under a separate heading for each function. The explanation of the functions covers only those points pertinent to the command interface. For full details, see the description of the dsl_ subroutine in Section 4 of this document. Notice that the dsl_ interface can return the sub_error_ condition (see the Note on sub_error_ in Section 4 and mrds_call set_modes function in this section).

Function: close, c

This function closes the specified data bases and makes them unavailable for further processing. The data base need not have been opened with the mrds_call command.

Usage

```
mrds_call close [data_base_index1 { ... data_base_indexn} | all_option]
```

where:

1. data_base_indexi
   is the data base index displayed by the open function.

2. all_option
   may be -all, or -a to specify that all of the user's open data bases are to be closed. This control argument may not be used with a data_base_index.

Example

The command line:

```
mrds_call close 1 2
```

closes data bases 1 and 2 regardless of how they were opened and makes them unavailable for further processing.

Function: declare, dcl

This function makes a user-defined function known to MRDS while processing the specified data base.

Usage

```
mrds_call declare data_base_index fn_name
```

where:

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

2. fn_name
   is the name of the function being declared.

Example

```
mrds_call declare 1 average
```

Function: define_temp_rel, dtr

This function creates, redefines, or deletes a temporary relation. The relation index corresponding to the temporary relation is displayed. This index (rel_index) must be specified in a later retrieve in order to reference the temporary relation. This may be done by a ".V." argument substitution of the rel_index in the range clause of the retrieve function.

The only functions that can be performed on a temporary relation are define_temp_rel, retrieve, and get_population.

Usage

```
mrds_call define_temp_rel data_base_index {selection_expression}
     {se_values} rel_index {-control_arg}
```

where:

1. data_base_index
   is the index displayed by the open function.

2. selection_expression
   is a character string as defined in Section 4, "Selection Mechanism." This argument must be omitted if the -segment control argument is specified.

3. se_values
   is a selection expression value (none, one, or more) for each control code (designated by .V.) appearing in the <selection_expression>. These must correspond in order and quantity with the control codes specified in the <selection_expression>.

4. rel_index
   is an integer. If equal to zero, a new temporary relation is created. If greater than zero, the temporary relation with that index is redefined. If less than zero, the temporary relation with that index is deleted.

5. control_arg
   may be -segment path or -sm path to specify that the selection expression is to be taken from the designated segment. (Refer to the examples included with the modify and retrieve functions described later in this section.)

Notes

For shared openings, read_attr scope must have been set on the referenced relations.

For attribute level security, attributes referenced in the where and select clauses must have at least read_attr access.

Unpopulated temporary relations can be created if the selection expression does not select any tuples. The get_population function can display the resulting tuple count.

Example

```
! mrds_call define_temp_rel 1 "-range (x phone_book)
!    -select x.name* x.mail_drop" 0

   Temporary relation index is: 1.
```

results in a new temporary relation being created with a relation index of 1.

*

Function: delete, dl

This function deletes selected tuples from the designated data base.

Usage

```
mrds_call delete data_base_index {selection_expression}
     {se_values} {-control_arg}
```

where:

1. data_base_index
   is the index displayed by the open function.

2. selection_expression
   is a character string as defined in Section 4, "Selection Mechanism". This argument must be omitted if the -segment control argument is specified.

3. se_values
   is a selection expression value (none, one, or more) for each control code (designated by .V.) appearing in the <selection_expression>. These must correspond in order and quantity with the control codes specified in the <selection_expression>.

4. control_arg
   may be -segment path, or -sm path to specify that the selection expression is to be taken from the designated segment. (Refer to the examples included with the modify and retrieve functions described later in this section.)

Notes

For shared openings, delete_tuple scope must have been set on the relation.

For attribute level security, the relation requires delete_tuple access, and any attributes referenced in the where clause require read_attr access.

Example

The command line:

```
mrds_call delete 1 "-range (x phone_book) -select x
   -where x.name = ""Smith, Roger D."" "
```

deletes the phone book entry associated with the name Smith, Roger D.

Function: dl_scope, ds

This function is used only with shared openings obtained with an opening mode of update or retrieval. Its purpose is to delete part or all of the current concurrency control scope modes on a relation basis. All scope must have been deleted from all relations before another scope setting operation can be accomplished.

Usage

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

where:

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

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

3. permit_scope_I
   is the set of operations that the user wishes to delete from the current permit scope for this relation. See the table of scope mode abbreviations below.

mrds_call

4. prevent_scope_I
is the set of operations that the user wishes to delete from the current prevent scope for this relation. See the table of scope mode abbreviations below.

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 of the above also applies to the prevent scope. Note that "n", does not delete any scope from that prevent or permit for the given relation.

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

Scope may be deleted entirely for all relations at once by using the dl_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 among processes requesting concurrent access protection.

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_settings

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

   Opening index:  1
          mode:  update
```

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

           Relation       Permits    Prevents

           rel1             ramd       n
           rel2             r          amd


  !  mrds_call  dl_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
                   data base model path: >udd>m>jg>dr>two_rels.db
                      data base version: 4

           Relation       Permits    Prevents

           rel1             r          n
           rel2             r          n
```

Function: dl_scope_all, dsa

This function deletes all scope from the user's current view of the data base.

Usage

```
mrds_call dl_scope_all data_base_index
```

where data_base_index is the data base index displayed by the open function.

Note

No error will be issued if there is no scope set when this function is used.

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 its 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

Since temporary relations do not store duplicates, it is not possible to get a true count of a selection expression tuple population where the -dup option is involved, unless temporary relation keys are defined over uniquely identifying attributes.

This function does not work for version 3 data bases.

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

```
! 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
```

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
                     data base model path: >udd>m>jg>dr>dmdm.db
                        data base version: 4

           Relation       Permits   Prevents

           sample           ramd      ramd


! mrds_call  get_scope  1  sample

   Permits:  ramd      Prevents:  ramd
```

Function: list_dbs, ld

For all openings of MRDS data bases in the user's process, this function displays the opening index, opening mode, and path of the submodel or model used for the opening.

Usage

```
mrds_call  list_dbs
```

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
```

Notes

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

Function: modify, m

This function causes the designated data base to be modified as specified.

Usage

```
mrds_call modify data_base_index {selection_expression}
     {se_values} modified_values {-control_arg}
```

where:

1. data_base_index
   is the index displayed by the open function.

2. selection_expression
   is a character string as defined in Section 4, "Examples of Selection Mechanisms". This argument must be omitted if the -segment control argument is specified.

3. se_values
   is a selection expression value (none, one, or more) for each control code (designated by .V.) appearing in the <selection_expression>. These must correspond in order and quantity with the control codes specified in the <selection_expression>.

4. modified_values
   is one or more values that are to replace the selected tuple attribute values in the data base.

5. control_arg
   may be -segment path or -sm path to specify that the selection expression is to be taken from the designated segment.

Notes

For shared openings, the relation must have modify_attr scope set.

For attribute level security, the selected attributes must have modify_attr

access and any attributes appearing in the where clause must have read_attr access.

Example

Assume the segment named mod_select contains:

```
-range (x phone_book)
-select x.phone
-where x.name = .V.
```

```
! mrds_call modify 1 "Jones, James A." 993-3064 -sm mod_select
```

changes the phone number of "Jones, James A." to 993-3064 (the se_value in this case). (Refer to the "Note" concerning the use of quotation marks included in the examples of the retrieve function described later in this section.)

*

Function: open, o

This function causes a data base to be opened and readied for use. It accepts either a model or submodel path for the opening and, in the default case, displays a data base opening index that is needed by other mrds_call functions.

Usage

```
mrds_call open view_path1 open_mode1 {...  view_pathN open_modeN}
```

where:

1. view_pathi
   is the pathname of the desired view to be used for this opening. This view can be either the path of the data base itself or the path of a submodel referring to the data base. The pathname can be relative or absolute and does not require any suffix, unless needed to prevent ambiguity. 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. open_modei
   is the desired opening mode for this opening of the data base. The following opening modes are available.

   retrieval, r
   specifies that this is a shared opening, requiring the setting of concurrency control protection via scope requests by the set_scope

function. The maximum permit scope that can be set with this opening mode is read_attr.

update, u
specifies that this is a shared opening, requiring the setting of concurrency control protection via scope requests by the set_scope function. Any scope can be set with this opening mode.

exclusive_retrieval, er
specifies that this is an unshared opening in the sense that all update operations are prevented against any relations in this view of the data base. 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 permit of read_attr and prevents of modify_attr, append_tuple, and delete_tuple on these relations. Other data base openers are allowed to set read_attr scope and do retrievals on these relations.

exclusive_update, eu
specifies that this is an unshared opening, in the sense that any operation is prevented by another user against any relation in this view of the data base. No scope setting is necessary with this opening mode. No other data base openers are allowed to set any scope on any relation in this view of the data base. 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 opener's 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 eliminated with the mrds_call set_modes no_list feature.

If the data base being opened has been secured, then the view_path must refer to a submodel that resides in the data base's "secure.submodels" directory under the data base directory if the user is not a DBA. These must be version 5 submodels if attribute level security is to be provided. See secure_mrds_db and Section 7 "Security".

If the data base 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 include "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 data base concurrency control segment. Unshared opening modes require that, for any relation appearing in the view, the multisegment 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; 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

The following example is for a non-DBA or a secured data base.

```
! secure_mrds_db  model display

The data base 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 data base 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 data base ">udd>m>jg>dr>model.db"
that has been secured, but the submodel itself is not in the data base's
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
```

The following example is for a non-DBA on an unsecured data base.

```
! secure_mrds_db  model  -display

The data base 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
                    data base model path:  >udd>m>jg>dr>model.db
                       data base version:  4

            Relation                            Permits   Prevents

            sample                                r          amd

   Opening index:  2
           mode:  exclusive_retrieval

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

            Relation                            Permits   Prevents

            sample                                r          amd

   Opening index:  3
           mode:  update

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

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

            No scope currently set for this opening.
```

Function: retrieve, r

This function retrieves and displays selected information from a data base.

Usage

```
mrds_call retrieve nvals data_base_index {selection_expression} {se_values}
     {-control_args}
```

where:

1. nvals
   is a decimal integer greater than zero specifying the number of attributes to be retrieved from the selected tuple.

2. data_base_index
   is the index displayed by the open function.

3. selection_expression
   is a character string as defined in "Examples of Selection Mechanisms" in Section 4. This argument must be omitted if the -segment control argument is specified.

4. se_values
   is a selection expression value (none, one, or more) for each control code (designated by .V.) appearing in the <selection_expression>, including temporary relation (rel_index) designations. These must correspond in order and quantity with the control codes specified in the <selection_expression>.

5. control_args
   may be one or both of the following:

   -all, -a
   specifies that all selected tuples be printed. If not specified, only the first selected tuple is printed and any subsequent tuples must be explicitly retrieved by a new retrieve function using "-another" for the selection expression.

   -segment path, -sm path
   specifies that the selection expression is to be taken from the designated segment (see "Notes" below).

Notes

The selection expression and the -segment control argument are mutually exclusive. If selection expression is specified, that argument becomes the selection expression for the retrieval. If the -segment control argument is specified, the selection expression is taken from the segment designated by path.

For shared openings, read_attr scope must have been set on any relations appearing in the range clause.

For attribute level security, read_attr access is required for attributes appearing in the select or where clause.

Examples

Assume the segment named query contains:

```
-range (x phone_book)
-select x.name x.mail_drop
-where x.phone = .V.
```

```
! mrds_call retrieve 2 1 993-3065 -all -segment query
```

```
     Values are:

     Jones, James A.
     B-116

     ******

     Smith, Roger D.
     B-116

     (END)
```

Both Jones and Smith have the specified phone number and are therefore selected.

```
  !  mrds_call retrieve 2 1 "-range (x phone_book)
  !    -select x.name x.mail_drop
  !    -where x.phone = ""993-3065"""

     Values are:

     Jones, James A.
     B-116
```

The command above did not use "-all" as a control argument, so the second value "Smith, Roger D. B-116" is not printed. This second value remains available to the user if a second retrieve function with a selection expression of "-another" is invoked prior to executing a retrieve function where the selection expression consists of an <alpha expression> (see "Formal Syntax").

```
  !  mrds_call retrieve 2 1 -another

     Values are:

     Smith, Roger D.
     B-116
```

results in the printing of the next value selected by the previous retrieve function's selection expression where the -all control argument is not used.

NOTE: The selection expression contained in the segment named query (first example) and -another (third example) are not contained in quotes. The selection expression in the second example specifies an argument in the mrds_call command line and is contained in quotes. The literal phone number value must be double quoted.

If a retrieve function is performed on a temporary relation, then the rel_index must be specified as a se_value. The command line:

```
mrds_call retrieve 2 1 "-range (x .V.)
    -select x" 6 -all
```

retrieves information from the temporary relation with the rel_index "6".

NOTE: Only a temporary relation index, not a relation name, may be used as a substitution value for the ".V." argument in the range clause.

*

mrds_call

Function: set_modes, sm

This function allows the user to control the amount of error or display information returned by mrds_call.

Usage

```
mrds_call set_modes {options}
```

where options may be either or both of (a) and (b) below:

(a) long_err to allow output from the sub_error_ condition or short_err to suppress it.

(b) list to allow the opening information to be displayed after an open function, or no_list to suppress it.

Note

If the set_modes function is not used, the default mrds_call action is long_err and list.

Function: set_scope, ss

This function is used only with shared openings obtained by using the opening modes of retrieval or update. Its 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 data base. 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 data base.

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 data base 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 user's process will wait before failing an attempt to set scope modes that conflict with another user's 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 (and prevent) 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".

Scope settings can be displayed by the get_scope function or by 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 can 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 among processes requesting concurrent access protection.

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

REQUESTED PERMIT	RELATION MSF ACL	MRDS ACCESS (Secure Data Bases Only)
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_settings

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

   Opening index:  1
            mode:  update

             Concurrency control version:  5
                     data base model path:  >udd>m>jg>dr>two_rels.db
                        data base 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).
```

mrds_call

Function: set_scope_all, ssa

This function is used only with shared openings obtained by using the opening modes of retrieval or update. Its 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 data base. The concurrency control modes, or scopes, are set on all relations at once.

Usage

mrc ssa 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 data base.

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 data base 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 user's process will wait before failing an attempt to set scope modes that conflict with another user's 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

mrds_call

permit scope. The abbreviation "u" is the same as specifying a permit scope of "amd". All of the above also applies to the prevent scope.

Scope settings can be displayed by the get_scope function or by 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 can 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 among processes requesting concurrent access protection.

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

REQUESTED PERMIT	RELATION MSF ACL	MRDS ACCESS (Secure Data Bases Only)
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_settings

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

Opening index:  1
         mode:  update

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

Relation	Permits	Prevents
rel1	ra	md
rel2	ra	md

Function: store, s

This function adds a specified tuple to the designated relation.

Usage

mrc s data_base_index relation_expression new_values

where:

1. data_base_index
   is the index displayed by the open function.

2. relation_expression
   indicates the relation to which a tuple is to be added. It may be the name of a relation or it may be "-another".

3. new_values
   are attribute values to be added to the new tuple.

Example

```
mrds_call store 1 phone_book "Newperson, John J."
  Engineering B-116 993-3062
```

results in the entry associated with Newperson being added to the phone book.

The following example references step 3c under "MRDS Tutorial" in Section 2:

```
mrds_call store 1 Comp_mgr Mfg 51603
mrds_call store 1 Employee Akins 57111 Eng
.
.
.
```

Notes

If an incomplete tuple is being stored (i.e., a tuple with one or more unknown attribute values), the user must select null values for inclusion in the tuple to prevent shifting of attribute values within domains/attributes. A suggestion is to enter a blank (" ") in attributes requiring alphabetic data and

a "-1" (or some type of numeric value that cannot be confused with valid data) for an attribute requiring numeric data.

Primary key attributes with null values in the key should never be entered in a data base.

If the relation_expression is the name of a relation, the new tuple is added to the named relation. If the relation_expression is "-another", the new tuple is added to the relation specified in the most recent invocation of store in which the relation_expression parameter consisted of a relation name. The user of any mrds_call command requiring a <selection_expression> causes the previously specified relation name to become unavailable for subsequent reference using "-another", until it is again established through the use of a mrds_call store function with a relation_expression consisting of the relation name.

The use of "-another" provides an efficient means to store several tuples into a single relation via consecutive mrds_call store functions.
For shared openings, append_tuple scope must have been set on the relation.

For attribute level security, the relation must have append_tuple access.

SECTION 10

OBSOLETE INTERFACES

This section is obsolete and has been deleted from the manual.

SECTION 11

CHANGES IN MRDS

This section is obsolete and has been deleted from the manual.

SECTION 12

EFFECT OF DATA BASE VERSION ON COMMANDS AND SUBROUTINES

This section is obsolete and has been deleted from the manual.

SECTION 13

PERFORMANCE CONSIDERATIONS

The following discussion on performance is divided into the following areas:

- Data base creation
- Data base use
- Selection expressions

These areas are not completely independent; the size and number of relations in a data base will affect the format of the selection expressions as will the number of times a data base is opened and the use of temporary relations. The DBA and the user must also be concerned with performance versus storage and performance versus maintainability considerations.

DATA BASE CREATION

For best retrieval performance, an attribute that is used to select tuples (i.e., appears in a where clause) should be either a secondary index, part of the key head (see key head access methods below), or a primary key. If the attribute is always used with some other attribute that is a secondary index, part of the key head, or the primary key, then this rule does not apply. An example might be latitude and longitude.

Indexing an attribute increases storage requirements for the relations by the length of the attribute plus 2 words for key overhead for each tuple in the relation. In addition, the time to do an update (store, modify, delete) operation increases slightly for each indexed attribute in the relation (it is independent of the number of tuples in the relation). However, the time it takes to select a set of tuples based on a condition on an indexed attribute is reduced tremendously -- from a linear function of the number of tuples in the relation to a logarithmic function.

A normalized data base (see "Data Base Design" in Section 2) will, in general, require fewer operations to do an update. Multiple operations may be done via one call to dsl_, (e.g., modifying all occurrences of 249-7790 to 249-8861) but may require more joins to select a tuple subset. While the join operation will probably be slower than performing the multiple update operations and will definitely cause a more complex selection expression to be used, it is felt that the advantages obtained by normalization, which are the removal of the update anomalies and the removal of duplicated data, outweigh its disadvantages.

Attribute values are encoded whenever the attribute is indexed or part of the primary key. This is done so that the data value and the bit pattern representing the data value have the same ordering. This is not the usual case, since the bit pattern of a negative number (fixed bin data type) is larger than any positive number because of the sign bit. Data types of character (N), where N is some integer, do not require any encoding. Data types of fixed bin (N,P) aligned, where N and P are integers, require minimal encoding (changing the sign

bit). All other data types require more complex encoding schemes. Attribute values and constants may need to be converted to other data types for comparison with other attributes or constants. The data types character (N) nonvarying and fixed (N, 0) are compared most efficiently.

Data Base Use

*

If a small portion of a relation is frequently accessed, a temporary relation defined over the tuple subset can improve retrieval speed. For example, suppose there are multiple queries (population, voter registration, per capita income, etc.) about the cities in a given state. Rather than composing selection expressions that include the state name, it is faster to create a temporary relation composed of just cities from the given state. The amount of performance improvement depends on the size of the base relation (total number of cities), the percentage of tuples that are always being considered (cities in the given state versus size of base relation), and the number of queries. The process of defining the temporary relation does require that the temporary directory have enough quota to hold the relation. This technique can also be used when the tuple subset comes from, or at least depends on, more than one relation so that the number of join operations is reduced.

Calling the dsl_ entries store, retrieve, modify, and delete with a long argument list does not incur the cost of breaking a structure down into its components and is, therefore, more efficient than making a call with a structure.

To avoid the cost of data conversion, the data type of the arguments in the calls to dsl_ store, retrieve, modify, and delete should match the attribute data type that they correspond to. To reduce conversion of the constants used in the selection expression, a ".V." may be used in place of the constant and an argument of the correct data type and value placed in the argument list.

A submodel opening has no performance effect after the opening phase. The opening phase may be faster or slower depending on the number of relations in the submodel view versus the number of relations in the data base. If the data base contains substantially more relations than the submodel view, a submodel view opening will be faster.

Because of the increased checking that must be done, operations on a secure data base are less efficient than the same operations performed on an unsecured data base.

SELECTION EXPRESSION

To minimize the cost of data movement, the minimum attributes needed should be selected from the tuples in the range (i.e., if a tuple has 13 attributes and only 7 are actually going to be used by the caller, it is faster to select only those 7 attributes rather than all 13).

When all attributes in a tuple are to be selected, it is more efficient to use the tuple variable name in the select clause rather than individually specifying each attribute.

To avoid the high cost of duplicate processing, the -dup option may be used. However, this option should only be used where the user can be certain that duplicate tuples will not occur or will not be a problem. Simple cases where duplicates cannot occur will have -dup forced by MRDS. These cases are limited to a single tuple variable where the entire key is selected.

Wherever possible, it is desirable to compare attributes to constant values rather than other attributes. For example,

```
TV1.key = 5 & TV2.key > TV1.key
```

is less efficient than

```
TV1.key = 5 & TV2.key > 5
```

Explicitly stating transitive conditions is also less efficient than leaving the condition implicit, not only because of the smaller number of terms in the where clause but because it prevents optimization. For example,

```
TV1.key = TV2.key & TV2.key = TV3.key
 & TV3.key = TV1.key
```

is less efficient than

```
TV1.key = TV2.key & TV2.key = TV3.key
```

It should be noted that the use of expressions, functions, and set operations is extremely slow.

The following is a list of the most to least efficient methods of accessing a relation for relations with a large number of tuples. For relations with a small number of tuples it is faster to do sequential searches because of the reduced overhead. The exact number of a large or small number of tuples depends on the opening mode, the number of duplicate secondary index values, and the number of attributes in the primary key. That is, it depends on the selectivity of the secondary indices and key heads.

Primary Key Equality	MOST
Key Head Equality	
Key or Key Head Range	
Indexed Attribute	
Sequential Search	LEAST

- Primary Key Equality implies that all the key attributes of the tuple variable (TV1) are equated to either a constant or to some other attribute in another tuple variable (TV2) whose value can be determined before the value of TV1.
- Key Head Equality implies that from 1 to N-1 of the tuple variable's (TV1) N key attributes are equated to either a constant or to some other attribute in another tuple variable (TV2) whose value can be determined before the value of TV1. In addition, the N-1 attributes must comprise a key head, that is the attributes must be the first N-1 attributes of the primary key, the order of attributes being determined by their order in the relation as defined by the cmdb source.
- Key or Key Head Range implies that a condition other than equality is being applied to the first key attribute of the tuple variable.
- Indexed Attribute implies that a condition is being applied to some indexed attribute.

- Sequential Search is used when the tuples in the tuple variable must be searched sequentially.

*

In cases where more than 1 access method may be used, the most efficient is chosen.

All where clause expressions are converted into disjunctive normal form as the first step in processing. Where clause expressions that are already in disjunctive normal form do not need to be converted and are, therefore, processed faster.

A where clause expression in disjunctive normal form has the form:

```
A ¦ B ¦ C ¦ D ¦ E ...
```

where each A, B, C, D, E, ... may contain any number of terms, but the terms must all be and'ed (&) together (an AND-GROUP). Each term must have the form:

```
(tup_var_Y.attr  rel_op  tup_var_X.attr)
```

Example

The expression:

```
^(TV1.at1 = 200) &
   ((TV1.at2 > TV2.at1) ¦ (TV1.at2 > TV2.at4))
```

is not in disjunctive normal form. To convert into that form the AND (&) operator must be distributed,[1] creating the expression:

```
(TV1.at1 = 200) & (TV1.at2 > TV2.at1)
                ¦
(TV1.at1 = 200) & (TV1.at2 > TV2.at4)
```

The expression:

```
(TV1.at1 = 200) &
   ^((TV1.at2 > TV2.at1) ¦ (TV1.at2 > TV2.at4))
```

is not in disjunctive normal form. To convert into that form, DeMorgan's rules may be applied to remove the NOT (^) operator from the expression.[2] Notice that the OR operator is removed without extra effort and the sense of the relational operators is reversed.

```
(TV1.at1 = 200) & (TV1.at2 <= TV2.at1) &
   (TV1.at2 <= TV2.at4)
```

MRDS constructs what it considers to be an optimum order for searching each tuple variable within each AND-GROUP. It does not optimize for more than one AND-GROUP at a time. In the first example (above) there would be two searches of tuple variable TV1 both looking for tuples where at1 equals 200. The best performance is therefore achieved if the where clause expression contains only one AND-GROUP.

---

[1] A & (B ¦ C) ==> A&B ¦ A&C

[2] ^(A ¦ B) ==> ^A & ^B

MRDS estimates the number of tuples that each tuple variable in an AND-GROUP (see disjunctive normal form) will select when generating the tuple variable search order for an AND-GROUP. These estimates may not be valid and may result in a search order that is not optimum. The major reason for an invalid estimate is a term that selects a disproportionately large or small number of tuples (i.e., the actual number of tuples selected depends on the data and cannot be determined without actually looking at the data), which defeats the purpose of the optimization. Note that the value of the estimate is not really important; only its magnitude when compared to the other estimates for the terms in the AND-GROUP is important. As long as their relative magnitudes are correct, the estimates will generate an optimum search order.

The -no_ot option of the selection expression allows the user to tell MRDS the order in which the tuple variables should be searched. This option coupled with the -print_search_order option allows the user to experiment to find a search order better than the one MRDS would generate. It has, however, several drawbacks, the most important of which is that it prevents the search order from changing due to changes in the data base content. Another is that only one search order can be defined and it will be applied to all the AND-GROUPS in the where clause expression. A very subtle drawback is the ability to select (without checking) what appears to be the obviously correct (but in fact incorrect) search order to save the time that MRDS would spend determining a search order. Search order determination is not obvious and should be approached with care.

Example 1

Given two relations rel_A and rel_B with the same number of tuples (say 1000) and the selection expression:

```
-range (A rel_A) (B rel_B)
 -select A B
  -where A.key_attr = B.non_key_attr
```

There are two possible search orders:

1. Search tuple variable A sequentially. For each tuple in A, search tuple variable B sequentially for a non_key_attr that equals the key_attr in A. This requires that the 1000 tuples in rel_B be searched for each tuple in relation rel_A for a total of 1000 * 1000 searches.

2. Search tuple variable B sequentially. For each tuple in B, use the primary key equality access method to find a tuple in tuple variable A. This requires that each tuple in rel_A and rel_B be searched only once for a total of 1000 + 1000 searches.

Obviously the second search order is superior.

Example 2

Casual inspection does not always immediately reveal the optimal search path. If the where clause in the previous example was changed to:

```
-where  A.key_attr = B.indexed_attr
```

then the search order "B first then A" is faster.

However, if the size of relation A is changed to 10 tuples, the costs of the searches become:

A before B - - - 10*(cost of index search of B)

B before A - - - 1000*(cost of key search of A)

Even for a key search cost several times smaller than an index search, the search order "A before B" may be the faster access method.

The cost of finding an optimal search order for an AND-GROUP is a factorial function of the number of tuple variables in that AND-GROUP (all possible orderings of the tuple variables are examined). If the where clause expression contains only one AND-GROUP or all the AND-GROUPS have the same search orders, then processing can be speeded up by using the -no_ot option. The search order may be determined by executing the selection expression once with the -print_search_order option.

From time to time the selection expression should be executed with the -print_search_order option but without the -no_ot option to be sure that the search order is still optimal. The time interval between these executions will depend on the volatility of the data base and must be judged on an individual basis by the user.

Note that the search order for an AND-GROUP that contains tuple variables defined over relations that are empty or have only a few tuples in them (compared to the other relations involved in the AND-GROUP) may change drastically as those relations are loaded with more tuples.

The -print_search_order selection expression option will cause the tuple variable search order for each AND-GROUP (see disjunctive normal form) in the where clause expression to be displayed. The display is output over the user_output switch. Each tuple variable is numbered; number 1 is searched first, 2 second, etc. Each AND-GROUP is separate and the tuple variable numbering starts over at 1. The display for each tuple variable contains:

1. the tuple variable name

2. the relation name the tuple variable is defined over

3. the access method

4. an estimate of the number of tuples selected from the tuple variable (not displayed to a non-DBA using a secure data base)

5a. the relational operator(s) and the attribute into which it is applied for the access methods Index Attribute and Key or Key Head Range (not displayed to a non-DBA using a secure data base)

5b. the number of key attributes for the access methods Primary Key Equality and Key Head Equality (not displayed to a non-DBA using a secure data base)

For the case where all the tuple variables are searched sequentially, a header to that effect is output along with each tuple variable name, its relation name, and the relation size.

If a tuple variable (TV1) occurs in the select clause but does not occur in an AND-GROUP (see disjunctive normal form) in the where clause, then when that AND-GROUP is processed, a cross product between that tuple variable (TV1) and the tuple variables in the select clause that have conditions in the AND-GROUP will be done. This implies two things:

1. TV1 is searched sequentially

2. the number of tuples returned will be S*N,

where S is the number of tuples that would be selected if TV1 were not in the select clause and N is the number of tuples in the relation TV1 refers to.

When there is no where clause, a cross product is formed between all the tuple variables in the select clause resulting in:

nTV1 * nTV2 * ... * nTVM

tuples retrieved, where nTVi is the number of tuples in the relation that tuple variable TVi refers to.

Note that the actual number of tuples retrieved may be smaller if duplicate processing is being done.

SECTION 14

RESTRUCTURING SUBSYSTEM

The Restructuring Subsystem is a facility available to the DBA to perform certain restructuring operations on MRDS data bases. The currently supported restructuring operations are the creation of new:

- attributes
- domains
- relations (and optional population)
- secondary indexes
- data base models

the deletion of existing:

- attributes
- domains
- relations
- secondary indexes

and the renaming of existing:

- attributes
- domains
- relations

To invoke the Restructuring Subsystem, the DBA must use the restructure_mrds_db (rmdb) command. The rmdb subsystem is an interactive subsystem that uses the standard subsystem utility package (ssu_). It supports certain features that are common to other subsystems such as abbrev processing, help request, the exec_com request, etc. There are also, quite naturally, requests that are specific to the rmdb subsystem.

Before a data base can be restructured, it must be quiesced by the rmdb facility (see Note). This is accomplished by supplying the data base pathname to the rmdb command line or explicitly by the ready_db request within rmdb. Similarly, exiting the subsystem with the quit request causes the data base to be unquiesced, and makes a free_db request available for explicit unquiescing.

Note: If the user has quiesced the data base (using the quiesce_mrds_db command) prior to entering the restructure_mrds_db subsystem, it uses the quiescent data base, and leaves it quiesced upon exit.

When restructuring takes place, history of the restructuring operation is retained in the model. This restructuring history can be displayed by using the -history control argument to the display_mrds_dm command at Multics command level. There is also an rmdb request that is a request-level interface to the display_mrds_dm command.

Although the subsystem is "interactive," it is possible under certain conditions for a restructuring operation to take a considerable amount of time. Given that, it is possible for a

restructuring operation to be interrupted before completion, leaving the data base in an inconsistent state. For this reason, when a restructuring operation begins, a flag in the model is set marking the data base as inconsistent. That flag is not reset until the restructuring operation is completed. In addition, a textual reason for inconsistency is saved (e.g., "Creating index IndA in relation RelB"). Further, an rmdb request, a "make-consistent operation", is saved. In the example stated it would be "delete_index RelB IndA". In some cases, the rmdb request may be null (since executing the request over again would result in the same inconsistency).

If a data base is left in such a state, and a user attempts to open it, an error message stating the reason for inconsistency is displayed and the user is directed to contact the DBA. The DBA must invoke the rmdb subsystem and access the data base. Upon determining that the data base is inconsistent, rmdb queries the DBA whether or not the make-consistent operation should be executed on the user's behalf. A positive response makes the data base consistent.

Name: restructure_mrds_db, rmdb

This command is used to enter the MRDS Restructuring Subsystem to restructure a given data base (see Notes below). If the data base does not exist it can be created. If the data base exists, and is not already quiesced, then it is quiesced.

Usage

rmdb {db_path} {-control_args}

where:

1. db_path
   is a relative or absolute path to the data base to be restructured.

2. control_args
   can be chosen from the following:

   -abbrev, -ab
   enables abbreviation expansion and editing of request lines.

   -force, -fc
   specifies that the data base be created if it does not already exist without querying the user.

   -no_abbrev, -nab
   suppresses the abbreviation expansion and editing of request lines. (Default)

   -no_force, nfc
   queries the user if the data base does not exist, to determine if the data base should be created. This argument overides the -force control argument. (Default)

   -no_prompt, -npmt
   suppresses the prompt in the request loop.

   -pathname db_path, -pn db_path
   specifies the path of the data base used for restructuring. The indicated data base is quiesced. This overrides any previously indicated data bases given via the optional db_path argument (above), or another -pathname control argument.

   -profile path, -pf path
   specifies the pathname of the profile used for abbreviation expansion. The profile suffix is added if necessary. This control argument implies -abbrev.

   -prompt STR, -pmt STR
   sets the request loop prompt to STR. (Default is "rmdb:")

   -quiesce_wait_time N, -qwt N
   sets the number of seconds that an attempt to quiesce waits for conflicting data base users to depart before failing. (Default is 0, that is, no waiting before failing.)

restructure_mrds_db

-relation_type type {modes}, -rt type {modes}
specifies the type of relation to create if the data base does NOT already exist. The supported types are vfile or data_management_file (dmf). The mode argument is only valid for dmf relations, and the supported modes are any combination of protected, concurrency, or rollback separated by commas. Any mode may be preceded with a NOT sign (^) to negate it. (Also see Notes below.)

-request STR, -rq STR
executes STR as an rmdb request line before entering the request loop.

-temp_dir path, -td path
provides the path of a directory that has more quota than the default of the process directory when more temporary storage is needed to restructure a large data base. If the user gets a record quota overflow in the process directory during an rmdb invocation, then a new_proc is required. A retry of the rmdb invocation with the -temp_dir argument, giving a pathname of a directory with more quota than the process directory, can then be done.

Notes

This command can only be used against a Version 4 or later data base and only by the DBA. In addition, this command cannot be used against a data base that is already open by any process. The data base can be opened (only by the process invoking this subsystem) after the subsystem is entered by invoking linus or the mrds_call command via the ".." (or execute) request.

If a new data base is to be created, and the -relation_type control argument is not specified, then the default relation type is vfile.

Restructure Requests

The following list summarizes all of the restructuring requests.

.
identifies rmdb with the version number and the pathname of the data base being restructured.

?
lists the available rmdb requests and active requests.

abbrev, ab
turns abbreviation processing ON or OFF and changes profile segments.

answer
supplies an answer to a question asked by a request.

create_attribute, cra
creates a new attribute based upon a previously defined domain. The attribute is unreferenced until it is used in a relation.

create_domain, crd
creates a new domain. A newly created domain is considered unreferenced

although it has a corresponding attribute of the same name defined upon itself.

create_index, cri
makes the indicated attribute a secondary index into the relation.

create_relation, crr
creates a new relation. An unpopulated relation can be specified by listing the attributes that make up the relation; each attribute must already be defined.

delete_attribute, dla
deletes the indicated attribute from the data base. The attribute is removed from all relations in which it is referenced.

delete_domain, dld
deletes the indicated domain from the data base. All attributes based upon the domain are also deleted causing restructuring of relations referencing those attributes.

delete_index, dli
deletes the secondary index over the indicated attribute in the relation.

delete_relation, dlr
deletes the indicated relation from the data base.

display_data_model, ddm, dmdm
displays details of the data base model.

do
substitutes args into the request_line and passes the result to the rmdb request processor.

exec_com, ec
executes the rmdb exec_com indicated by ec_path. The ec_path arguments are passed to the exec_com processor.

execute, e
executes a Multics command line after evaluating rmdb active requests.

free_db, fdb
unquiesces the data base.

help
displays information about request names or topics. A list of available topics is produced by the list_help request.

if
conditionally executes a request.

list_help, lh
displays a list of available info segments whose names include a topic string.

list_requests, lr
displays information about rmdb requests.

quit, q
restores the current data base to a non-quiescent state (if the current data base was quiesced by the rmdb subsystem) and leaves rmdb.

restructure_mrds_db

ready_db, rdb
quiesces the indicated data base and makes it available for restructuring. Note that only one data base can be restructured at any given time. If the data base does NOT exist, a query is made to determine if an empty data base is to be created.

rename_attribute, rna
renames the indicated attribute.

rename_domain, rnd
renames the indicated domain and its corresponding attributes.

rename_relation, rnr
renames the indicated relation.

subsystem_name
displays the name of the subsystem, "rmdb".

subsystem_version
displays the current version of rmdb.

The remainder of this section contains a detailed description of each request, including standard subsystem environmental requests, that is, requests common to other subsystems such as abbrev, answer, do, etc.

Request: .

This request identifies rmdb with the Version number and the path of the data base being restructured.

Usage

.

Example

```
!.
 rmdb 1.0:     >udd>Demo>mrds>...
```

Request: ?

This request displays the available restructure_mrds_db requests.

Usage

?

Example

The following list is displayed when "?" is entered by the user to the prompt "rmdb:".

```
rmdb:  ?
rmdb:  Available rmdb requests:

    .                       display_data_model,     exec_com, ec
    create_attribute,          ddm, dmdm            execute, e
       cra                  free_db, fdb            execute_string, exs
    create_domain, crd      ready_db, rdb           help
    create_index, cri       rename_attribute,       if
    create_relation, crr       rna                  list_help, lh
    delete_attribute,       rename_domain, rnd      list_requests, lr
       dla                  rename_relation, rnr    quit, q
    delete_domain, dld      abbrev, ab              substitute_arguments,
    delete_index, dli       answer                     substitute_args,
    delete_relation, dlr    do                         sbag

Type "list_requests" for a short description of the requests.
```

---

restructure_mrds_db

Request: abbrev, ab

This request controls abbreviation processing within the subsystem. As an active request, it returns "true" if abbreviation expansion of request lines is currently enabled within the subsystem and "false" otherwise.

Usage

ab {-control_args}

Usage as an Active Request

[ab]

where control_args can be chosen from the following (and cannot be used with the active request):

-off
specifies that abbreviations are not to be expanded.

-on
specifies that abbreviations should be expanded. (Default)

-profile path
specifies that the segment named by path is to be used as the profile segment; the profile suffix is added to path if not present. The segment named by pat h must exist.

Notes

This subsystem provides command line control arguments (-abbrev, -no_abbrev, -profile) to specify the initial state of abbreviation processing within the subsystem. For example, a Multics abbreviation can be defined to invoke the read_mail subsystem with a default profile as follows:

.ab rdm do "read_mail -abbrev -profile [hd]>mail_system &rf1"

If invoked with no arguments, this request enables abbreviation processing within the subsystem using the profile that was last used in this subsystem invocation. If abbreviation processing was not previously enabled, the profile in use at Multics command level is used; this profile is normally [home_dir]>Person_id.profile.

See the abbrev command in the Multics Commands for a description of abbreviation processing.

Request: answer

This request provides preset answers to questions asked by another request.

Usage

answer STR {-control_args} request_line

where:

1. STR
   is the desired answer to any question. If the answer is more than one word, it must be enclosed in quotes. If STR is -query, the question is passed on to the user. The -query control argument is the only one that can be used in place of STR.

2. request_line
   is any subsystem request line. It can contain any number of separate arguments (i.e., have spaces within it) and need not be enclosed in quotes.

3. control_args
   can be chosen from the following:

   -brief, -bf
   suppresses display (on user terminal) of both the question and the answer.

   -call STR
   evaluates the active string STR to obtain the next answer in a sequence. The active string is constructed from subsystem active requests and Multics active strings (using the subsystem "execute" active request). The outermost level of brackets must be omitted (i.e., "forum_list -changed") and the entire string must be enclosed in quotes if it contains request processor special characters. The return value "true" is translated to "yes," and "false" to "no." All other return values are passed as is.

   -exclude STR, -ex STR
   passes on, to the user or other handler, questions whose text matches STR. If STR is surrounded by slashes (/), it is interpreted as a qedx regular expression. Otherwise, answer tests whether STR is literally contained in the text of the question. Multiple occurrences of -match and -exclude are allowed (see "Notes" below). They apply to the entire request line.

   -match STR
   answers only questions whose text matches STR. If STR is surrounded by slashes (/), it is interpreted as a qedx regular expression. Otherwise, answer tests whether STR is literally contained in the text of the question. Multiple occurrences of -match and -exclude are allowed (see "Notes" below). They apply to the entire request line.

   -query
   skips the next answer in a sequence, passing the question on to the user. The answer is read from the user_i/o I/O switch.

   -then STR
   supplies the next answer in a sequence.

   -times N
   gives the previous answer (STR, -then STR, or -query) N times only (where N is an integer).

restructure_mrds_db

Notes

The answer request provides preset responses to questions by establishing an ON unit for the condition command_question and then executes the designated request line. If any request in the request line calls the command_query_ subroutine (described in the Multics Subroutines) to ask a question, the ON unit is invoked to supply the answer. The ON unit is reverted when the answer request returns to subsystem request level. See "List of System Conditions and Default Handlers" in the Reference Manual for a discussion of the command_question condition.

If a question is asked that requires a yes or no answer, and the preset answer is neither "yes" or "no," the ON unit is not invoked.

The last answer specified is issued as many times as necessary, unless followed by the -times N control argument.

The -match and -exclude control arguments are applied in the order specified. Each -match causes a given question to be answered if it matches STR; each -exclude causes it to be passed on if it matches STR. A question excluded by the -exclude control argument is reconsidered if it matches a -match later in the request line. For example, the request line:

```
answer yes -match /fortran/ -exclude /fortran_io/ -match /^fortran_io/
```

answers questions containing the string "fortran", except that it does not answer questions containing "fortran_io". It does, however, answer questions beginning with "fortran_io".

Request: create_attribute, cra

This request creates an unreferenced attribute in the currently readied data base.

Usage

```
cra attribute1 domain1 {...attributeN domainN}
```

where:

1. attributei
   is the name of the attribute to be created.

2. domaini
   is the name of the underlying domain. The domain must already exist.

Request: create_domain, crd

This request creates an unreferenced domain in the currently readied data base.

restructure_mrds_db

Usage

crd domain_name data_type {-control_args}

where:

1. domain_name
   is the name of the domain to be created.

2. data_type
   is the underlying data type of the domain. If the data_type contains spaces or parentheses, it MUST be quoted. See "Notes" for a list of supported data types.

3. control_args
   can be chosen from the following:

   -check_procedure path, -check_proc path
   performs data verification checks (such as ensuring valid dates) upon storage into the data base. "path" may be an absolute or relative pathname.

   -decode_declare data_type, -decode_dcl data_type
   is the underlying data type of the argument to the decode procedure for this domain. See "Notes" for a list of supported data types.

   -decode_procedure path, -decode_proc path
   performs data decoding upon retrieval from the data base, normally the inverse of the encode procedure. "path" may be an absolute or relative pathname.

   -encode_procedure path, -encode_proc path
   performs data encoding (such as the names of the states of the USA to integers 1-50) before storage in an internal data base form. "path" may be an absolute or relative pathname.

Notes

Any legal PL/1 scalar data type that can be declared using the following declaration description words is allowed in MRDS.

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

Request: create_index, cri

This request creates a secondary index for the attribute in the relation.

restructure_mrds_db

Usage

cri relation_name attribute_name

where:

1. relation_name
   is the name of the relation to be restructured.

2. attribute_name
   is the name of the attribute to be indexed.

Request: create_relation, crr

This request creates a new relation in a data base.

Usage

crr relation_name {rel_attribute_list} {-control_args}

where:

1. relation_name
   is the name of the relation to be created.

2. rel_attribute_list
   is a list of the attribute names used in the relation. The rel_attribute_list has the syntax of attr_1 attr_2 ... attr_n (where "attr_"s are the attribute names of the attributes to be used for the relation). The attribute names that are to make up the primary key of the relation must have an appended "*". The rel_attribute_list cannot be used if the -selection_exp control argument is provided.

3. control_args
   can be chosen from the following:

   -index STR, -ix STR
   specifies the list of attributes in the relation that are indexed. STR has the syntax of attr_1 attr_2 ... attr_n (where "attr_"s are the attribute names of the attributes to be indexed). If the -selection_exp control argument is used, the -index control argument must precede the -selection_exp control argument.

   -selection_exp STR {select_values}, -se STR {select_values}
   STR is a selection expression that defines relation attributes that are to be created and populated using the data selected by the selection expression. See "help mrds.selection_expressions" for the define_temp_rel selection expression specification. The selection expression must be a separately quoted string with any select_values provided as individual arguments. The -selection_exp control argument, if provided, must be the last control argument.

---

restructure_mrds_db

Request: delete_attribute, dla

This request deletes referenced or unreferenced attributes from a MRDS data base.

Usage

dla {attribute_name1 {...attribute_nameN} {-control_args}

where:

1. attribute_namei
   is the name of the attribute(s) to be deleted from the MRDS data base.

2. control_args
   can be chosen from the following:

   -all, -a
   deletes all attributes defined in the MRDS data base. This control argument is inconsistent with -check.

   -brief, -bf
   suppresses the -long display. (Default) The last occurrence of -brief and -long on the command line takes effect.

   -check, -ck
   prevents the deletion of any attributes selected during the execution of this command and, instead, traces all implied operations upon the data base and displays them on the terminal. This trace consists of a statement for each attribute that is referenced, listing the relations that reference the attribute.

   -force, -fc
   prevents the query from being issued if any of the attributes are referenced in the MRDS data base. (Default is to issue a separate query for each referenced attribute.)

   -inhibit_error, -ihe
   prevents error messages from being issued to the terminal. (Default is to issue error messages.)

   -long, -lg
   displays the same output as -check; however, the specified attributes are deleted.

   -no_force, -nfc
   overrides the -force control argument. The last occurrence of -force and -no_force on the request line takes effect. (Default)

   -no_inhibit_error, -nihe
   overrides the action of -inhibit_error. (Default)

   -unreferenced, -unref
   deletes only unreferenced attributes. This control argument overrides -all and is inconsistent with -check.

restructure_mrds_db

Notes

If an attribute is referenced in one or more relations, ripple effects take place. When the attributes are actually deleted, all relations that use the deleted attributes are reformatted.

Specifying either -all or -unreferenced and a list of domain names on the request line is flagged as an inconsistent error.

A query is issued for each referenced attribute that is to be deleted to ensure against catastrophic data loss. With the -long control argument, the query is of the form:

Attribute "start_date" is used in relations "permanent_employees" and "temporary_employees". Do you wish to delete the attribute start_date?

Request: delete_domain, dld

This request deletes the specified domains from a MRDS data base. The domains may be referenced or unreferenced.

Usage

dld {domain1 {...domainN} {-control_args}

where:

1. domaini
   are the domains to be deleted.

2. control_args
   can be chosen from the following:

   -all, -a
   deletes all domains defined in the MRDS data base. This control argument is inconsistent with -check.

   -brief, -bf
   suppresses the trace display. (Default) The last occurrence of -brief and -long on the command line takes effect. This argument is inconsistent with -check.

   -check, -ck
   prevents the deletion of any domains selected during the execution of this command, and instead, traces all implied operations upon the data base and displays them on the terminal. This trace consists of a statement for each domain that is referenced, listing the domain that is to be deleted, a list of attributes that are based upon the domain, and a list of all relations that are to be modified. Inconsistent with -brief or -long.

   -force, -fc
   prevents the query from being issued for domains which are referenced in

the MRDS data base. (Default is to issue a separate query for each referenced domain.)

-inhibit_error, -ihe
prevents error messages from being issued to the terminal. (Default is to issue error messages.)

-long, -lg
displays the same output as -check; however, the specified domains are deleted. The last occurrence of -brief and -long on the command line takes effect. This control argument is inconsistent with -check.

-no_force, -nfc
overrides the -force control argument. The last occurrence of -force and -no_force on the command line takes effect. (Default)

-no_inhibit_error, -nihe
overrides the action of -inhibit_error. (Default)

-unreferenced, -unref
deletes only unreferenced domains. This control argument is inconsistent with -check and -all.

Notes

If the domain is referenced in attributes, which are themselves referenced in relations, ripple effects take place. When the domains are actually deleted, all attributes based upon them are also deleted. This causes the relations that use the deleted attributes to be reformatted.

Specifying either -all or -unreferenced and a list of domain names on the request line is flagged as an inconsistent error.

A query is issued for each referenced domain that is to be deleted to ensure against catastrophic data loss. The query is of the form:

Domain clock_value is used in attributes "clock_value", "start_date", "stop_date", and "current_date" which are referenced in relations "permanent_employees" and "temporary_employees". Do you wish to delete it?

Request: delete_index, dli

This request removes the secondary index for the attribute in the relation.

Usage

dli relation_name attribute_name {-control_args}

where:

restructure_mrds_db

1. relation_name
   is the name of the relation to be restructured.

2. attribute_name
   is the name of the attribute whose secondary index is to be deleted.

3. control_args
   can be chosen from the following:

   -brief, -bf
   suppresses error reporting if the attribute is not already a secondary index.

   -long, -lg
   reports an error if the attribute is not already a secondary index. (Default)

Request: delete_relation, dlr

This request deletes a relation from the data base.

Usage

dlr relation_name {-control_args}

where:

1. relation_name
   is the name of the relation to be deleted.

2. control_args
   can be chosen from the following:

   -brief, -bf
   specifies that no errors are reported.

   -long, -lg
   specifies that errors are reported. (Default)

Request: display_data_model, ddm, dmdm

This request displays the model definition of a MRDS data base, including domain, attribute, and relation information.

Usage

ddm {-control_args}

or:

restructure_mrds_db

dmdm {-control_args}

where control_args can be chosen from the following:

-attribute {modifier}, -attr {modifier}
displays attribute information. The modifier may be name(s) or -unreferenced (-unref). If name(s) is supplied, information for the attribute name(s) is displayed. If -unreferenced is supplied, attribute information about all unreferenced attributes is displayed. If no modifier is supplied, attribute information about all attributes is displayed.

-brief, -bf
specifies that the brief format be displayed. This argument is incompatible with -names.

-cmdb
specifies that the output be in the same format as an input source text for create_mrds_db. If the -output_file control argument is included in the invocation, then the segment can be used to create another data base with the same definitions. Only the -brief, -long, and -output_file control arguments can be used with this control argument.

-crossref {type}, -xref {type}
displays an information cross-reference. The type may be domain (dom), attribute (attr), or all. If the type is domain, each domain is listed with a list of attributes in which the domain is referenced. If the type is attribute, each attribute is listed with a list of relations in which the attribute is referenced. If the type is all, both domain and attribute cross-references are displayed. (Default is "all".) See the examples below which show the information displayed.

-domain {modifier}, -dom {modifier}
displays domain information. The modifier may be name(s) or -unreferenced (-unref). If name(s) is supplied, information for the domain name(s) is displayed. If -unreferenced is supplied, domain information about all unreferenced domains is displayed. If no modifier is supplied, domain information about all domains is displayed.

-header, -he
displays data base header information.

-history, -hist
displays restructuring history information. If the data base is restructured more then once, the history entries are displayed in reverse chronological order.

-index names, -ix names
displays information about indexed relations for the relation names supplied. If no names are supplied, then information about all indexed relations is displayed.

-long, -lg
specifies that the long format be displayed. This argument is incompatible with -names.

-names, -nm
displays the format of domains, attributes, relations, and indexed relations as a list of the names. This argument is incompatible with -brief or -long.

restructure_mrds_db

-no_header, -nhe
prevents display of the header information. (Default)

-no_output_file, -nof
writes the output to the terminal. (Default)

-output_file path, -of path
writes the output to path, rather than to the terminal.

-relation names, -rel names
displays relation information for the relation names supplied. If no names are supplied, the relation information about all relations is displayed.

-temp_dir path
specifies that the directory indicated by path be used for temporary storage.

Note

If no control arguments are supplied, the default relation information is displayed.

Examples

If the data base "little" is created from the source:

```
domain:  code fixed bin, address char(20);
relation:  zip(code* address);
```

the results would be as follows:

```
display_data_model -long
DATA MODEL FOR DATA BASE  >udd>Demo>dbmt>db7>jg>little.db

Version:                4
Created by:          User.Multics.a
Created on:          05/14/80  1042.9 mst Wed

Total Domains:          2
Total Attributes:       2
Total Relations:        1

RELATION NAME:   zip

Number attributes:        2
Key length (bits):       36
Data Length (bits):     216

   ATTRIBUTES:

      Name:       code
      Type:       Key
      Offset:     0 (bits)
      Length:     36 (bits)
      Domain_info:
```

restructure_mrds_db

```
            name: code
            dcl:  real fixed binary (17,0) aligned

      Name:       address
      Type:       Data
      Offset:     36 (bits)
      Length:     180 (bits)
      Domain_info:
            name: address
            dcl:  character (20) nonvarying unaligned


display_data_model -cmdb -long
  .
  .
  .
 /* Created from   >udd>Demo>dbmt>db7>User>little.db
                    06/14/82  1251.3 mst Wed      */

domain:
          address
               character (20) nonvarying unaligned,
          code
               real fixed binary (17,0) aligned;

relation:
          zip                  (code* address);
```

Request: do

This request expands a request line by substituting the supplied arguments into the line before execution. As an active request, it returns the expanded request_string rather than executing it.

Usage

```
do request_line {args}
```

or:

```
do -control_args
```

Usage as an Active Request:

```
[do request_line args]
```

where:

1. request_line
   is a request line in quotes.

restructure_mrds_db

2. args
   are character string arguments that replace parameters in request_string.

3. control_args
   can be chosen from the following to set the mode of operation:

   -long, -lg
   displays the expanded request line before execution.

   -brief, -bf
   specifies that the expanded request line not be printed before execution. (Default)

   -nogo
   specifies that the expanded request line not be passed on for execution.

   -go
   specifies that the expanded request line be passed on for execution. (Default)

   -absentee
   establishes an any_other handler that catches all conditions and aborts execution of the request line without aborting the process.

   -interactive
   specifies that the any_other handler not be established. (Default)

List of Parameters

Any sequence beginning with & in the request line is expanded by the do request using the arguments given on the request line.

&I
is replaced by argI. I must be a digit from 1 to 9.

&(I)
is replaced by argI. I may be any value.

&qI
is replaced by argI with any quotes in argI doubled. I must be a digit from 1 to 9.

&q(I)
is replaced by argI with any quotes in argI doubled. I may be any value.

&rI
is replaced by argI surrounded by level quotes with any contained quotes doubled. I must be a digit from 1 to 9.

&r(I)
is replaced by a requoted argI. I may be any value.

&fI
is replaced by all the arguments starting with argI. I must be a digit from 1 to 9.

&f(I)
is replaced by all the arguments starting with argI. I may be any value.

&qfI
is replaced by all the arguments starting with argI with any quotes doubled. I must be a digit from 1 to 9.

&qf(I)
is replaced by all the arguments starting with argI with quotes doubled. I may be any value.

&rI
is replaced by all the arguments starting with argI. Each argument is placed in level quotes with contained quotes doubled. I must be a digit from 1 to 9.

&rf(I)
is replaced by all the arguments starting with argI, requoted. I may be any value.

&&
is replaced by an ampersand.

&!
is replaced by a 15-character unique string. The string used is the same in every place where the &! appears in the request line.

&n
is replaced by the actual number of arguments supplied.

&f&n
is replaced by the last argument supplied.

Request: exec_com, ec

This request executes a program written in the exec_com language that is used to pass request lines to the subsystem and to pass input lines to requests that read input. As an active request, it specifies a return value by use of the &return statement.

Usage

ec ec_path {ec_args}

Usage as an Active Request

[ec ec_path {ec_args}]

where:

1. ec_path
is the pathname of an exec_com program. The suffix, which is normally the name of the subsystem, is assumed if not specified.

restructure_mrds_db

2. ec_args
are optional arguments to the exec_com program and are substituted for parameter references in the program such as &1.

Notes

Subsystems may define a search list to be used to find the exec_com program. If this is the case, the search list is used if ec_path does not contain a "<" or ">" character; if the ec_path contains either a "<" or ">", it is assumed to be a relative pathname.

For a description of the exec_com language (both Version 1 and Version 2), type:

.. help v1ec v2ec

When evaluating a subsystem exec_com program, subsystem active requests are used rather than Multics active functions to evaluate the &[...] construct and the active string in an &if statement. The execute active request of the subsystem can be used to evaluate Multics active strings within the exec_com.

Limitation: In the present implementation, any errors detected during execution of an exec_com within a subsystem aborts the request line in which the exec_com request is invoked.

Request: execute, e

This request executes the supplied line as a Multics command line after evaluating rmdb active requests. As an active request, it evaluates a Multics active string and returns the result to the subsystem request processor.

Usage

e STR

Usage as an Active Request

[e STR]

where STR is the Multics command line to be executed or the Multics active string to be evaluated. It need not be enclosed in quotes.

restructure_mrds_db

Notes

The recommended method to execute a Multics command line from within a subsystem is the ".." escape sequence. The execute request is intended as a means of passing information from the subsystem to the Multics command processor.

All (), [], and "s in the given line are processed by the subsystem request processor and not the Multics command processor. This permits passing values of subsystem active requests to Multics commands when using the execute request, or passing values to Multics active functions for further manipulation before returning the values to the subsystem request processor for use within a request line.

Examples

The rmdb request line:

```
exec_com [execute hd]>create_temp.rmdb
```

can be used to execute an rmdb exec_com in the user's home_directory.

The rmdb request line:

```
execute display_mrds_temp_dir -current
```

can be used to review the name of the directory that is being used by mrds for temporary storage.

Request: free_db, fdb

This request frees the data base currently readied by the Restructuring Subsystem from the subsystem (i.e., allows the data base to be opened by any user and prevents further restructuring requests against the data base).

Usage

```
fdb
```

Request: help

This request displays information about various subsystem topics including detailed descriptions of most subsystem requests.

restructure_mrds_db

Usage

help {topics} {-control_args}

where:

1. topics
   specifies the topics on which information is to be displayed. The topics available within a subsystem can be determined by using the list_help request if available.

2. control_args
   can be chosen from the following:

   -brief, -bf
   displays a summary of a request or active request, including the syntax, list of arguments, control arguments, etc.

   -search STRs, -srh STRs
   displays the paragraph containing all the strings identified by STRs. (Default, the display begins at the top of the information.)

   -section STRs, -scn STRs
   displays the section whose title contains all the strings identified by STRs. (Default, the display begins at the top of the information.)

   -title
   displays section titles and section line counts, then asks if the user wants to see the first paragraph of information.

List of Responses

The most useful responses that can be given to questions asked by the help request are:

.
displays "help" to identify the current interactive environment.

.. command_line
treats the remainder of the response as a Multics command line.

?
displays a list of responses allowed.

no, n
stops display of information and proceeds to the next topic, if any.

quit, q
stops display of information and returns to subsystem request level.

rest {-section}, r {-scn}
displays remaining information without intervening questions. If -section is given, help displays the rest of the current section, without questions, and then asks if the user wants to see the next section.

search {STRs} {-top}, srh {STRs} {-t}
skips to the next paragraph containing all the strings identified by STRs.

restructure_mrds_db

If -top is given, searching starts at the top of the information. If STRs are omitted, help uses the STRs from the previous search response, or the -search control argument.

section {STRs} {-top}, scn {STRs} {-t}
skips to the next section whose title contains all the strings identified by STRs. If -top is given, title searching starts at the top of the information. If STRs are omitted, help uses the STRs from the previous section response, or the -section control argument.

skip {-section}} {-seen}, s {-scn} {-seen}
skips to the next paragraph. If -section is given, the request skips all paragraphs of the current section. If -seen is given, the request skips to the next paragraph that the user has not seen. Only one control argument is allowed in each skip response.

title {-top}
displays titles and line counts of the sections that follow. If -top is given, help displays all section titles and repeats the previous question after titles are displayed.

yes, y
prints the next paragraph of information on this topic.

Notes

If no topic names are given, the help request explains what help requests are available in the subsystem.

restructure_mrds_db

For a complete description of the control arguments and responses accepted by this request, type:

help help

Request: if

This request conditionally executes one of two request lines depending on the value of an active string. As an active request, it returns one of two character strings to the subsystem request processor depending on the value of an active string.

Usage

if expr -then line1 {-else line2}

Usage as an Active Request

if expr -then STR1 {-else STR2}

where:

1. expr
   evaluates the active string as "true" or "false." The active string is constructed from subsystem active requests and Multics active strings (using the execute active request of the subsystem).

2. line1
   executes the subsystem request line if expr is "true." If the request line contains any request processor characters, it must be enclosed in quotes.

3. line2
   executes the subsystem request line if expr is "false." If omitted and expr is "false," no additional request line is executed. If the request line contains any request processor characters, it must be enclosed in quotes.

4. STR1
   returns this value to the active request when expr is "true."

5. STR2
   returns this value to the if active request when expr is "false." If omitted and the expr is "false," a null string is returned.

Request: list_help, lh

This request lists the names of all subsystem info segments pertaining to a given set of topics.

restructure_mrds_db

Usage

lh {topics}

where topics specifies the topics of interest. Any subsystem info segment that contains one of these topics as a substring is listed.

Notes

If no topics are given, all info segments available for the subsystem are displayed.

An info segment name is considered to match a topic only if that topic is at the beginning or end of a word within the segment name. Words in info segment names are bounded by the beginning and end of the segment name and by the period (.), hyphen (-), underscore (_), and dollar sign ($) characters. The info suffix is not considered when matching topics.

Examples

The request line:

list_help list

matches info segments named list, list_users, and forum_list, but does not match an info segment named prelisting.

Request: list_requests, lr

This request displays a brief description of selected subsystem requests.

Usage

lr {STRs} {-control_args}

where:

1. STRs
   specifies the requests to be displayed. Any request with a name containing one of these strings is displayed unless -exact is used, in which case the request name must match exactly one of these strings.

restructure_mrds_db

2. control_args
can be chosen from the following:

-all, -a
includes undocumented and unimplemented requests in the display of requests eligible for matching the STR arguments.

-exact
displays only those requests whose names match exactly one of the STR arguments.

Notes

If no STRs are given, all requests are displayed.

A request name is considered to match a STR only if that STR is at the beginning or end of a word within the request name. Words in request names are bounded by the beginning and end of the request name and by the period (.), hyphen (-), underscore (_), and dollar sign ($) characters.

Examples

The request line:

```
list_requests list
```

matches requests named list, list_users, and forum_list, but does not match a request named prelisting.

Request: quit, q

This request is used to exit the subsystem, unquiesce the data base (if the current data base was quiesced by the rmdb subsystem), and return to Multics command level.

Usage

```
q
```

Request: ready_db, rdb

This request readies a data base for restructuring.

restructure_mrds_db

Usage

rdb {db_path} {-control_args}

where:

1. db_path
   is the relative or absolute path for the data base to be restructured. The db suffix is assumed if not supplied.

2. control_args
   can be chosen from the following:

   -force, -fc
   specifies that the data base be created if it does not already exist without querying the user.

   -no_force, -nfc
   overrides the -force control argument. (Default) The last occurrence of -force and -no_force on the command line takes effect.

   -pathname db_path, -pn db_path
   specifies the path for the data base to be restructured. The last path supplied is the readied one.

   -quiesce_wait_time N, -qwt N
   specifies the number of seconds to wait for all open users to close the data base. (Default is 0)

   -relation_type type {modes}, -rt type {modes}
   specifies the type of relation to create if the data base does not already exist. The supported types are vfile_ and data_management_file (dmf) (see Notes below). The mode argument is only valid for dmf-type relations, and the supported modes are any combination of protection, concurrency, or rollback separated by commas. Any mode may be preceded with a not sign (^) to negate it.

Notes

DBAs are the only persons who can ready a data base for restructuring.

The data base should not be readied if there are any open users Once the data base is readied, it can be opened by the process that has readied it.

The db_path argument cannot refer to a submodel or a data base earlier than Version 4.

This request can be run only against a consistent data base. If the data base is inconsistent, the user is queried to see if he/she wishes to execute the "undo request" and make the data base consistent. After executing the undo request, the data base can be readied. If the undo request fails, the user is returned to rmdb request level (i.e., the data base is not readied).

restructure_mrds_db

When this request is used to create a new data base, and the -relation_type argument is not specified, the data base is created with the default relation type of vfile_.

Only one data base can be readied at any given time.

Request: rename_attribute, rna

This request replaces the name of an attribute with another name.

Usage

rna attribute1 name1 {...attributeN nameN}

where:

1. attributei
   specifies the current name of an existing attribute.

2. namei
   specifies the new name that replaces the original name.

Request: rename_domain, rnd

This request replaces the name of a domain with another name.

Usage

rnd domain1 name1 {...domainN nameN}

where:

1. domaini
   specifies the current name of an existing domain.

2. namei
   specifies the new name that replaces the original name.

Request: rename_relation, rnr

This request replaces the name of a relation with another name.

restructure_mrds_db

Usage

rnr relation1 name1 {...relationN nameN}

where:

1. relationi
   specifies the current name of an existing relation.

2. namei
   specifies the new name that replaces the original name.

Request: subsystem_name

This request displays the name of the subsystem. As an active request, it returns the name of the subsystem.

Usage

subsystem_name

Usage as an Active Request

[subsystem_name]

Request: subsystem_version

This request displays the version number of the subsystem. As an active request, it returns the version number of the subsystem.

Usage

subsystem_version

Usage as an Active Request

[subsystem_version]

SECTION 15

DATA MANAGEMENT SYSTEM INTERFACE

MRDS now supports two distinct types of data bases; vfile_ data bases and Data Management System (DMS) data bases. The vfile_ data bases are those data bases which have relations that are created and accessed via the vfile_ IO module. These were the only kind of data bases that existed prior to MR11. The DMS data bases are those data bases which have relations that are created and accessed via the DMS facility. (Refer to "Data Management Overview" in the Programmer's Reference Manual for additional information.) MRDS does not support a combination of vfile_ relations and DMS relations within a single data base.

CREATING A DATA BASE

Data bases are created in the traditional fashion (i.e., by use of the create_mrds_db command). The create_mrds_db command has two new control arguments (-vfile and -data_management_file). The -vfile control argument (Default) causes a new data base to be a vfile_ data base, whereas the -data_management_file control argument causes a new data base to be a DMS data base. So, a command line that created a data base prior to MR11 will do precisely the same thing in MR11, that is, create a vfile_ data base. Therefore, the user must explicitly request a DMS data base.

CONVERTING A DATA BASE

To convert an existing vfile_ data base to a DMS data base, the user must first create a new DMS data base using the same cmdb source used to create the vfile_ data base. If the cmdb source segment is not available, it can be created using the -cmdb control argument with the display_mrds_dm command. Once the new data base is created, the user can copy the data from the old data base using the copy_mrds_data command. After execution of the copy_mrds_data command, there are two data bases with identical data, one a vfile_ data base, the other a DMS data base. The user can delete the vfile_ data base or continue to use the vfile_ data base in production, while testing the DMS data base, whichever circumstances warrant. The user can always invoke the copy_mrds_data command to reverse the procedure (i.e., copy data from a DMS data base back to a vfile_ data base).

FEATURES

The DMS facility provides new services for the MRDS user. One important feature is the concept of a transaction. In order to access a DMS file (in MRDS, a relation), the user's process must have initiated a transaction. When a transaction is active, each change made to the file causes a "before" image of that data to be

written to a "before" journal. If the transaction does not successfully complete (e.g., the system crashes, the user's process dies, or the user explicitly requests the transaction to be aborted or rolled back), the "before" images are used to return the relation to the state it was in before the DMS changes began. If the transaction completes successfully, the "before" images are discarded and the changes are made permanent. This can be quite useful to the programmer or interactive user who wishes to "group" a series of physically separated actions into a single logical unit.

The DMS interface that handles transactions is the transaction_manager_ subroutine. The command and subroutine interfaces to transaction_manager_ that are of interest to the MRDS user are documented later in this section.

The burden of starting and finishing transactions is not completely placed on the user. When MRDS accesses a DMS file, it knows there must be a transaction running; therefore, it checks to see if one is in place. If there is a transaction in place, MRDS proceeds to call the DMS facility, knowing that either the user or application is controlling the transaction. If no transaction is running, MRDS starts one prior to calling DMS and then, upon return from DMS, either commits the transaction, rolls it back and tries again, or aborts the transaction and returns an error code to the user. An application that accesses a DMS data base, therefore, need never interface with transaction_manager_ at all, relying on MRDS to handle transactions. In this fashion, an application that runs against a vfile_ data base could be used, without modification, to run against a DMS data base. MRDS simply turns each dsl_ call into a transaction.

In order to access a DMS file, the user must have initiated a transaction and have a "before" journal (provided by system default). Users can, however, create and use their own "before" journals. In fact, the site administrator may choose to restrict access to the system default "before" journal and require users to create and use their own. The DMS interface that handles "before" journals is the before_journal_manager_ subroutine. The command and subroutine interfaces to before_journal_manager_ that are of interest to the MRDS user are documented later in this section.

CHOOSING BETWEEN DATA BASE TYPES

With two types of data bases available (vfile_ and DMS), a choice must be made as to which type to use for a given application. The differences are many. The strength of the DMS system is integrity and consistency. The ability to group several operations into a go/nogo set can be quite valuable. Also, DMS ensures the integrity of the data itself, except for certain media failures. As an example, when MRDS is adding a tuple to a vfile_ data base, it (actually vfile_relmgr_) must call vfile_ several times to add both data and indexes. While vfile_ can guarantee the integrity of each individual action requested of it, the various calls are not logically related. If the system crashes, or the user's process fails between calls, the process can easily end up with a tuple that is only partially there, that is, the data and its indexes are not completely in place. In the DMS case, however, the adding of a tuple is a single atomic operation to DMS and is done within the confines of a single transaction. If the system crashes, or the user's process fails, the transaction is aborted by the DMS Daemon.

MRDS and vfile_ is significantly more efficient than MRDS and DMS. The choice between vfile_ or DMS depends upon the need for the additional functionality provided by DMS. If DMS functionality is important to an application, then those features may well outweigh efficiency considerations. If, however, the improvements

offered by DMS are not important to an application, it may be best to leave it as a vfile_ data base.

The recommended method of investigating DMS options, at least as it affects existing data bases, is to use the procedures described above and do some testing. The user can see what the various advantages and disadvantages of DMS are without affecting production jobs. This procedure will provide a feel for the issues of transactions and "before" journals. The copy_mrds_data command itself can be instructive. By default, it creates a transaction for each tuple copied. A transaction, while reasonably efficient, does involve some overhead. This overhead, multiplied over many tuples, can be significant. For this reason, the copy_mrds_data command utilizes the -transaction_group_size control argument. The operand indicates how many tuples are to be copied in a single transaction. If this control argument is used, an increase in efficiency will be seen (by not starting and committing as many transactions), but the size required for the "before" journal is increased as there are more "before" images involved in a single transaction. These same issues become involved in determining the usage of transactions within an application.

DMS COMMAND AND SUBROUTINE DESCRIPTIONS

before_journal_status

Name: before_journal_status, bjst

bjst {PATHS} {-control_args}

FUNCTION

displays status information for before journals that you have access to open. This command is part of the command level interface to Multics data management (DM) (see the Programmer's Reference Manual).

ARGUMENTS

PATHS
: are the relative pathnames of before journals for which status is desired. If you supply no pathnames, status information for all journals in use in the process is displayed. If you don't give the .bj suffix, it is assumed.

CONTROL ARGUMENTS

-all
: displays the status of all journals active in the current invocation of the data management system (DMS) that you have access to open.

-brief, -bf
: displays the pathname, unique identifier, usage state or activity, control interval size, and control intervals in the before journal for each journal specified that is either in use or not in use (see "Examples").

-long, -lg
: for each journal specified that is in use, displays, besides the above information, the disposition of control intervals in use, i.e., if they are buffered, put, flushed, or on disk; the last time a control interval was queued or written; the time the header was updated; the last record id; the status of images not yet written on disk or not being flushed; and the number of users and transactions using the journal. For each journal specified that is not in use, displays, besides the information given by -brief, the time the header was updated. (See "Examples.")

NOTES

If you give neither -brief nor -long, the command yields the information supplied by -brief plus the disposition of control intervals in use at the time of the request if the journal(s) specified is in use.

before_journal_status

EXAMPLES

The example below requests the status, in long form, of the system_low system default before journal, which is in use.

```
!  bjst >site>dm>system_low>system_default -lg

pathname:                       >site>Data_Management>system_low
                                  >system_default.bj
journal uid:                    132233107561
activity:                       in use
control interval size:          4096 bytes
control intervals:              4000
control intervals used:         86
last control interval
          buffered:             86
          put:                  86
          flushed:              86
          on disk:              86
time last control interval
          queued:               01/14/85  1104.9 est
          written:              01/14/85  1104.9 est
time header updated:            01/14/85  1104.9 est
last record id:                 000001260013
images not on disk:             0
images being flushed:           0
users:                          2
transactions:                   1
```

where:

pathname
: is the pathname of the before journal.

journal uid
: is the octal unique identifier of the before journal.

activity
: is "in use" if a process currently has the before journal open, "not in use" otherwise.

control interval size
: is the size of each control interval in the before journal, in bytes. Currently 4096 bytes is the only supported size.

control intervals
: is the number of control intervals in the before journal.

before_journal_status

control intervals used
is the number of control intervals in the before journal containing before images still needed to roll back modifications made by a transaction. Images that are not needed include those that have already been used in a complete rollback and those for a transaction that has ended.

last control interval buffered
indicates the last control interval put in a special buffer used for before journals.

last control interval put
indicates the last control interval put into the before journal.

last control interval flushed
indicates the last control interval flushed to disk.

last control interval on disk
indicates the last control interval safely on disk.

time last control interval queued
is the last time a before image was put in the before journal.

time last control interval written
is the last time a control interval was written to disk.

time header updated
is the last time the header of the before journal was written.

last record id
is the address of the last before image in the journal.

images not on disk
is the number of images not written to disk yet.

images being flushed
is the number of before images for which a flush from memory to disk has been requested.

users
is the number of users with openings.

transactions
is the number of active transactions in the before journal.

The example below requests the status, in long form, of the system_low system default before journal, which is not in use.

```
! bjst >site>dm>system_low>system_default -lg

pathname:                    >site>dm>system_default.bj
journal uid:                 127120202215
activity:                    not in use
control interval size:       4096 bytes
control intervals:           4000
time header updated:         08/26/84  1228.6 edt
```

Name: bj_mgr_call, bjmc

SYNTAX AS A COMMAND

```
bjmc key {paths} {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[bjmc key {paths} {-control_args}]
```

FUNCTION

enables you to manipulate before journals in your process by calling before_journal_manager_ entry points from command level. This command is part of the command level interface to Multics data management (DM) (see the Programmer's Reference Manual).

ARGUMENTS

key
designates the before journal manager operation to be performed. See "List of Operations" below for a description of each operation, its command and active function syntax lines, and specific application.

paths
specifies the absolute or relative pathname of the before journals being manipulated (required for all key operations except get_default_path). Give -pathname (-pn) PATH with pathnames constructed with leading minus signs to distinguish them from control arguments. If you supply no .bj suffix, it is assumed.

CONTROL ARGUMENTS

can be one or more control arguments, depending on the particular operation.

LIST OF OPERATIONS

Each operation is described in the general format of a command/active function. Where appropriate, notes and examples are included for clarity.

```
close, cl
closed
create, cr
get_default_path, gdp
open, o
opened
set_default_path, sdp
set_attribute, sattr
```

Operation: close, cl

SYNTAX AS A COMMAND

```
bjmc cl .paths
```

SYNTAX AS AN ACTIVE FUNCTION

```
[bjmc cl paths]
```

FUNCTION

closes the before journals specified by paths. Separate pathnames by spaces if multiple journals are to be closed. Specifically close by name each journal opened in the process. The active function returns true if the journals were closed successfully, false otherwise.

ARGUMENTS

paths
: are the absolute or relative pathnames of before journals to be closed. You can use -pathname (-pn) to specify the journal paths. If you supply no .bj suffix, it is assumed.

NOTES

If a before journal being closed by this operation is the default journal, the last journal opened in the process becomes the default.

bj_mgr_call

Operation: closed

SYNTAX AS A COMMAND

```
bjmc closed path
```

SYNTAX AS AN ACTIVE FUNCTION

```
[bjmc closed path]
```

FUNCTION

returns true if the before journal specified by path is not open in your process, false otherwise.

ARGUMENTS

path
is the absolute or relative pathname of a before journal. You can use -pathname (-pn) to specify the journal path. If you don't give the .bj suffix, it is assumed.

Operation: create, cr

SYNTAX AS A COMMAND

```
bjmc cr paths {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[bjmc cr paths {-control_args}]
```

FUNCTION

creates the before journals specified by paths. The active function returns true if the journals are created successfully, false otherwise.

ARGUMENTS

paths
are the absolute or relative pathnames of the before journals to be created. You can use -pathname (-pn) to specify the journal path. If you supply no .bj suffix, it is assumed.

CONTROL ARGUMENTS

-length N, -ln N
specifies the size of the before journal, where N is the number of 4096-byte control intervals. Once established, you can't alter a journal's size. (Default: if you specify no value at the time of creation, the size is 64 control intervals).

-transaction_storage_limit N, -tsl N
specifies the maximum number of bytes a single transaction can use in the before journal (Default: the entire journal, see the set_attriubute operation for more info).

NOTES

Before journals are extended entry types; you can delete them using the delete command. You can only delete before journals if they are not required for recovery.

Operation: get__default__path, gdp

SYNTAX AS A COMMAND

```
bjmc gdp
```

SYNTAX AS AN ACTIVE FUNCTION

```
[bjmc gdp]
```

FUNCTION

returns the pathname of the process's default before journal.

Operation: open, o

SYNTAX AS A COMMAND

```
bjmc o paths
```

SYNTAX AS AN ACTIVE FUNCTION

```
[bjmc o paths]
```

FUNCTION

opens the before journals specified by paths. The active function returns true if the journals are opened successfully, false otherwise.

ARGUMENTS

paths
are the absolute or relative pathnames of before journals to be opened in your process. You can use -pathname (-pn) to specify the journal path. If you supply no .bj suffix, it is assumed.

bj_mgr_call

NOTES

If no journal has been specifically designated as the default (see the set_default_path operation) for your process, the last before journal opened in the process becomes the default. If no journal is opened in your process when a transaction is started, the system before journal is opened and used as the default.

Operation: opened

SYNTAX AS A COMMAND

```
bjmc opened path
```

SYNTAX AS AN ACTIVE FUNCTION

```
[bjmc opened path]
```

FUNCTION

returns true if the before journal specified by path is opened in your process, false otherwise.

ARGUMENTS

path
: is the absolute or relative pathname of a before journal. You can use -pathname (-pn) to specify the journal path. If you supply no .bj suffix, it is assumed.

Operation: set_default_path, sdp

SYNTAX AS A COMMAND

```
bjmc sdp path
```

SYNTAX AS AN ACTIVE FUNCTION

```
[bjmc sdp path]
```

FUNCTION

sets the default before journal for the process to the specified pathname. The active function returns true if the pathname is successfully set, false otherwise.

ARGUMENTS

path
is the absolute or relative pathname of the before journal to be used as the default by your process. You can use -pathname (-pn) to specify the journal path. If you supply no .bj suffix, it is assumed.

NOTES

If no default before journal is set for your process, the last journal opened in the process is used as the default (see the open operation). If no before journal is open in the process when a transaction is started, the system before journal is opened and used as the default.

Operation: set_attribute, sattr

SYNTAX AS A COMMAND

```
bjmc sattr paths -control_arg
```

SYNTAX AS AN ACTIVE FUNCTION

```
[bjmc sattr paths -control_arg]
```

FUNCTION

sets an attribute of the before journals specified by paths. The active function returns true if the attribute is successfully set, false otherwise.

ARGUMENTS

paths
are the absolute or relative pathnames of the before journal(s) to have attributes set. You can use -pathname (-pn) to specify the journal path. If you supply no .bj suffix, it is assumed.

CONTROL ARGUMENTS

-transaction_storage_limit N, -tsl N
specifies the maximum number of bytes a single transaction can use in the before journals. An attempt to write more bytes than allowed causes the transaction_bj_full_ condition. A value of zero indicates a transaction can use an entire journal (the default at journal creation time).

NOTES

When this operation completes, the before journal header containing the new attributes is not guaranteed to be flushed if the journal is active. Any changes do take effect immediately for current user of the journal.

transaction

Name: transaction, txn

SYNTAX AS A COMMAND

```
txn key {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[txn key {-control_args}]
```

FUNCTION

enables you to define and execute atomic operations interactively. You can invoke the services of the transaction manager to begin, commit, abort, rollback, abandon, or kill a transaction. There is also a status request for displaying information about the current transaction. There is an execute request to wrap a given command line in a transaction. This command is part of the command level interface to Multics data management (DM) (see the Programmer's Reference Manual).

ARGUMENTS

key
: designates the operation to be performed. See "List of Operations" below for a description of each operation, its command syntax line, and specific application.

CONTROL ARGUMENTS
: can be one or more control arguments, depending on the particular operation.

LIST OF OPERATIONS

Each operation is described in the general format of a command/active function. Where appropriate, notes and examples are included for clarity.

Operation: abandon

SYNTAX AS A COMMAND

```
txn abandon
```

SYNTAX AS AN ACTIVE FUNCTION

```
[txn abandon]
```

FUNCTION

your process surrenders control of the transaction to the DM Daemon, which aborts it as part of its normal caretaker responsibilities. The active function returns true if the transaction is successfully abandoned, false otherwise.

NOTES

By abandoning a transaction, your process can start another transaction without waiting for the abort operation to conclude (your process is still charged for the abort). The data locked by the original transaction remains inaccessible, however, until the rollback is completed.

Operation: abort

SYNTAX AS A COMMAND

```
txn abort
```

SYNTAX AS AN ACTIVE FUNCTION

```
[txn abort]
```

FUNCTION

aborts the current transaction so that, in effect, it never existed. Any modifications to protected files caused by the aborted transaction are rolled back, and references to the transaction are removed from system tables. The active function returns true if the transaction is successfully aborted, false otherwise.

Operation: begin

SYNTAX AS A COMMAND

```
txn begin {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[txn begin {-control_args}]
```

FUNCTION

starts a transaction by reserving a slot in the transaction definition table (TDT) for your process, with a unique transaction identifier, date/time of the start, pathname of the before journal, and other information pertinent to the transaction (see the status operation). If your process already owns a transaction, an error occurs. The active function returns true if a transaction is started successfully, false otherwise.

CONTROL ARGUMENTS

-no_wait, -nwt
causes an error if the data management system (DMS) is not currently invoked. (Default)

-wait N, -wt N
if DMS is not currently invoked, wait N seconds before starting the transaction. An error occurs if DMS is still not up after the elapsed time.

-wait_indefinitely, -wti
if DMS is not currently invoked, wait as long as necessary to start the transaction. The status of DMS is checked at 10-second intervals, and notification is given when command line execution begins.

NOTES

This operation is a tool for isolating and testing the transaction startup function. In a production environment the transaction execute command is the recommended method of starting transactions from command level because it builds in the atomicity: it begins the transaction, executes a command line, and then terminates the transaction, within the one request (see the execute operation).

EXAMPLES

The following example shows an absentee job intended not to run until a transaction can be started in absentee.

```
&if &[not [txn begin -wait 100]] &then &do
    ear &ec_path -time "+1 hour" -ag &f1
    &quit
&end
```

Operation: commit

SYNTAX AS A COMMAND

```
txn commit
```

SYNTAX AS AN ACTIVE FUNCTION

```
[txn commit]
```

FUNCTION

signals successful completion of the currently active transaction. Modifications made to protected files by this transaction are considered permanent. Any locks held by the transaction are released, making the data public again. The active function returns true if the commit operation is successful, false otherwise.

Operation: execute, e

SYNTAX AS A COMMAND

```
txn e {-control_args} {command_line}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[txn e {-control_args} {command_line}]
```

FUNCTION

starts a transaction, executes a command line, and, provided the command line is successfully executed, commits the transaction. Control arguments govern what action to take based on conditions encountered. The active function returns true if the execute operation is successful, false otherwise.

ARGUMENTS

command_line
specifies the command line to be executed as part of the transaction. Enclose it in quotes if it contains parentheses, brackets, or semicolons. If you omit it, the system prompts "Command line:".

CONTROL ARGUMENTS

-abandon_on CONDITION_LIST
abandons the transaction and results in a nonlocal exit of the command line if any of the listed conditions is encountered during command line execution. Separate the listed conditions by commas, with no intervening whitespace. The list can include any_other. The default action is as described under "Notes" below. This control argument is incompatible with -existing_transaction_allowed and -existing_transaction_required.

-abort_on CONDITION_LIST
aborts the transaction and results in a nonlocal exit of the command line if any of the listed conditions is encountered during command line execution. Separate the listed conditions by commas, with no intervening whitespace. The list can include any_other. The default action is as described under "Notes" below. This control argument is incompatible with -existing_transaction_allowed and -existing_transaction_required.

-command_level, -cl
places your process at the next command level, from which commands can be entered in the transaction. You can use the start or release command to exit this command level.

-existing_transaction_allowed, -eta
accepts the existing transaction (if one already exists in your process) as the origin of command line execution. No new transaction is begun. This control argument is incompatible with -retry_on and -suspend_on. (Default: to return an error if a transaction already exists)

-existing_transaction_required, -etr
requires that a transaction already exist in your process; returns an error if no transaction exists. This control argument is incompatible with -retry_on and -suspend_on. (Default: to return an error if a transaction already exists)

-no_action_on CONDITION_LIST
overrides any special action (e.g., -abandon_on, -retry_on) you previously specified in the command line for the listed conditions. The default action (see "Notes") is also overridden.

-no_existing_transaction_allowed, -neta
causes an error if a transaction already exists in your process. (Default)

-no_wait, -nwt
causes an error if DMS is not currently invoked. (Default)

-retry_on N CONDITION_LIST
executes the command line up to N times if any of the listed conditions is encountered during command line execution. If N is 0, the command line is not retried. Separate the listed conditions by commas, with no intervening whitespace. The list can include any_other. The default action is as described under "Notes" below.

-suspend_on CONDITION_LIST
suspends the transaction and goes to the next command level if any of the listed conditions is encountered during command line execution. Separate the listed conditions by commas, with no intervening whitespace. The list can include any_other. The default action is as described under "Notes" below.

-wait N, -wt N
if DMS is not currently invoked, waits N seconds before starting the transaction and executing the command line (you are notified when command line execution begins). An error condition is returned if DMS is still not up after the elapsed time. This operation is useful for absentee jobs submitted to perform operations within transactions.

-wait_indefinitely, -wti
if DMS is not currently invoked, waits as long as necessary to start the transaction and execute the command line. The status of DMS is checked at 10-second intervals, and notification is given when command line execution begins.

NOTES

If a transaction already exists in your process, the default action is -no_action_on any_other; otherwise the default action is -suspend_on any_other -abort_on cleanup.

A transaction begun by txn execute is committed unless the command line fails to execute properly, in which case the transaction is aborted.

A transaction severity code (displayable by the "severity transaction" command) denotes the status of the execute operation, as follows:

0 the operation was completed without errors and was not retried.
1 the operation was completed, but was retried one or more times.
2 the operation failed; the transaction was aborted or abandoned.
3 the operation failed; the transaction could not be aborted or abandoned.
4 the transaction could not be begun.

The active function returns true if the severity after execution is 0 or 1; false if it is 2, 3, or 4.

If a transaction is currently suspended in your process, the txn execute command gets an error and the active function returns false.

Operation: kill

SYNTAX AS A COMMAND

```
txn kill {ID}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[txn kill {ID}]
```

FUNCTION

expunges the current or specified transaction with no attempt to preserve consistency of any DM files that might have been modified by this transaction. Killing a transaction may destroy the consistency of any databases that the transaction is using; therefore use this operation when neither you nor the Daemon is able to complete the transaction. The active function returns true if the operation is executed successfully, false otherwise.

ARGUMENTS

ID
is the unique identifier of the transaction to be killed (obtainable through txn status). (Default: the current transaction in your process)

ACCESS REQUIRED

You need re access to dm_daemon_gate_.

Operation: rollback

SYNTAX AS A COMMAND

```
txn rollback
```

SYNTAX AS AN ACTIVE FUNCTION

```
[txn rollback]
```

FUNCTION

rolls back the current transaction to its beginning (txn begin), undoing any changes to protected files caused by the transaction and releasing the locks held by it. The transaction is still considered active in your process. The active function returns true if the transaction was successfully rolled back, false otherwise.

Operation: status, st

SYNTAX AS A COMMAND

```
txn st {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[txn st {-control_args}]
```

FUNCTION

displays information about the current transaction, selected transactions, or all transactions, depending on the nature of the request and your access permissions. The active function takes only one information control argument.

CONTROL ARGUMENTS FOR SELECTING TRANSACTIONS

If you supply no control arguments, or lack the proper access, only information pertaining to your current transaction is displayed.

-abandoned
    displays the requested information about all TDT entries marked as abandoned.

-all, -a
    displays the requested information about all TDT entries.

-dead
displays the requested information about all TDT entries belonging to dead processes.

-transaction_id ID, -tid ID, -id ID
displays the requested information about the transaction with unique identifier ID, where ID is a decimal integer. Transaction identifiers are assigned at txn begin time and can be viewed by the txn status command.

-transaction_index N, -tix N, -index N
displays the requested information about entry number N in the TDT. TDT entry indexes are of interest mainly to data management maintainers and can be viewed by the txn status command.

CONTROL ARGUMENTS FOR SELECTING INFORMATION

If you give none of the following control arguments, all information is displayed for each TDT entry selected. You can specify only one control argument for the active function.

-before_journal_path, -bj_path
returns the pathname of the before journal used by the current transaction.

-date_time_begun, -dtbg, -begun
returns the date and starting time of each transaction.

-error, error_info
returns a description of the latest error, if any, to have occurred while processing each transaction.

-owner
identifies the owner (User_id.Project_id) of each TDT entry.

-process_id, -pid
returns the octal process_id of the owner of each TDT entry.

-rollback_count, -rbc
returns the number of times each transaction has been rolled back.

-state
indicates the state of each transaction, which must be one of the following:

no transaction (e.g., the process might have owned a transaction, which has been taken over by the DM Daemon)

in progress

{Error - } OPERATION, calling PROGRAM_NAME, which gives the operation currently in progress, such as abort or commit, and the entry point being called; and is followed by an error message if appropriate.

-switches, -switch, -sw
lists those transactions that are either abandoned, killed, or suspended or whose owner processes are dead.

-total, -tt
prints totals information for the TDT, including:

number of slots available (not yet reserved by processes)

number in use (i.e., reserved by processes at first invocation of DMS)

number of entries owned by dead processes (of the number in use)

number of abandoned entries (of the number in use)

number of entries occupied by transactions (i.e., slots reserved by processes that have started transactions)

number of transactions in error.

-transaction_id, -tid, -id
supplies the unique identifier of each transaction. Use of -transaction_id with a specific transaction ID returns information about that transaction.

-transaction_index, -tix, -index
returns the index of entries in the TDT. This index is mainly of interest to data management maintainers. Use of -transaction_index with a specific integer N returns information about a given TDT entry number.

NOTES

You can't use the following control arguments with the active function: -abandoned, -all, -dead, and -total.

You need re access to dm_admin_gate_ to view the status of any other user's transactions.

EXAMPLES

The command

```
! txn status -tid
  9
```

asks for the unique identifier of the transaction currently owned by the requesting user process.

The command

```
! txn status -a -owner -dtbg -tid
  TDT size: 6 entries
  In use: 4
  Dead processes: 1
  Abandoned entries: 0
  Transactions: 3
  Error transactions: 0

  Transaction id: 4
  Owner: Merrill.Multdev
  Begun at: 02/12/84 0837.11 est wed

  Owner: Lynch.Multdev
  No Transaction.

  Transaction id: 9
  Owner: Pierce.Debug
  Begun at: 02/12/84 0846.3 est wed

  Transaction id: 12
  Owner: Fenner.Support
  Begun at: 02/12/84 0901.5 est wed
```

requests that each transaction in the TDT be identified as to its unique identifier, owner, and date/time of origin.

The command

```
! txn status
  Transaction id: 4
  TDT index: 2
  Process id: 467265315627
  Owner: Smith.Applications
  Begun at: 02/12/84 0846.3 est wed
  State: In progress
  Error: none
  Checkpoint id: 0
  Rollback count: 0
  Before journal path: >site>dm>system_low>system_default.bj
  Switches: none
```

requests all available information on the transaction owned by the requesting user process.

The command

```
! txn status -tix 1 -pid -state -error -switches
   Process id: 625731253642 (dead)
   State: Error - Abort, calling bjm_$write_aborted_mark
   Error: The before journal is full.
   Switches: ABANDONED, DEAD_PROCESS
```

requests the process id, state, error condition, and switch settings for the specified transaction index entry.

Name: before_journal_manager_

The before_journal_manager_ subroutine provides the means to manipulate, and obtain information about, before journals. Before journals are used to store before images of protected data management (DM) files, for the purpose of rolling back modifications to these files in the event of failure.

See the section entitled "Multics Data Management" in the *Programmer's Reference Manual,* Order No. AG91, for a complete description of before journals and their use.

Entry: before_journal_manager_$close_bj

This entry point closes the specified before journal, making it unavailable to the current process. A journal can be opened more than once in a process, in which case the same opening id is returned for each open request. In that case, the close operation merely decreases by one the number of journal openings in the process. If a close_bj request is issued by a process on a journal while the process still has an active transaction in that journal, the journal cannot be closed and an error code is returned to the caller. If the journal to be closed was the default before journal for the process, the before journal which was last opened in the process (if any) becomes the default before journal (see "Notes" under the set_default_bj entry).

USAGE

```
declare before_journal_manager_$close_bj entry (bit(36) aligned, fixed
      bin(35));

call before_journal_manager_$close_bj (bj_opening_id, code);
```

ARGUMENTS

bj_opening_id
   is the opening identifier of the before journal. (Input)

code
is a standard system error code. (Output)

Entry: before_journal_manager_$create_bj

This entry point creates a before journal file as specified by the input arguments.

USAGE

```
declare before_journal_manager_$create_bj entry (char(*), char(*), fixed
      bin, fixed bin, fixed bin(35));

call before_journal_manager_$create_bj (dir_name, entry_name,
      n_control_intervals, control_interval_size, code);
```

ARGUMENTS

dir_name
is the pathname of the directory in which the before journal is to be created. (Input)

entry_name
is the entry name of the before journal to be created. The .bj suffix must be included. (Input)

n_control_intervals
is the size of the journal expressed in the number of control intervals. (Input) A before journal is a circular file; when information is no longer useful (i.e., before images for committed or aborted transactions), it will be overwritten, allowing the space to be reused. In estimating the size of a journal, you should consider the number of transactions to be using the journal simultaneously, as well as their profiles, i.e., their length in time and the rate at which they modify data, to optimize performance.

control_interval_size
is the size of the before journal control interval in number of bytes. (Input) The size is currently fixed at 4096.

code
is a standard system error code. (Output)

Entry: before_journal_manager_$get_default_bj

This entry point returns the opening identifier of the before journal to be used as the default in those cases where a before journal specification is expected but not supplied. The rules for determining this default before journal are described in "Notes" under the set_default_bj entry point. If the journal which is to serve as the default before journal is not open at the time of this call, it is opened automatically.

USAGE

```
declare before_journal_manager_$get_default_bj entry (bit(36) aligned,
      fixed bin(35));

call before_journal_manager_$get_default_bj (bj_oid, code);
```

ARGUMENTS

bj_oid
    is the opening identifier of the current default before journal. (Output)

code
    is a standard system error code. (Output)

Entry: before_journal_manager_$open_bj

This entry point makes the before journal specified by the pathname, ready for use by any transaction of the current process. A process may have several before journals open at the same time, and may also have the same journal opened more than one time. When a transaction is started, one of the open journals must be associated with the transaction, if the transaction needs a before journal. One can expect that in most cases, a process will open only one before journal, which will be used by all its transactions.

This entry may also change the default before journal for the process to the newly opened journal (see "Notes" under set_default_bj).

USAGE

```
declare before_journal_manager_$open_bj entry (char(*), char(*), bit(36)
      aligned, fixed bin(35));

call before_journal_manager_$open_bj (dir_name, entry_name,
      bj_opening_id, code);
```

ARGUMENTS

dir_name
    is the pathname of the directory in which the before journal to be opened resides. (Input)

entry_name
is the entry name of the before journal to be opened. The .bj suffix must be included (Input)

bj_opening_id
is the opening identifier of the journal. (Output) This specifier must be used subsequently by the current process to identify this journal.

code
is a standard system error code. (Output)

NOTES

When a before journal is opened, it is remembered in a per system table containing the pathnames and unique identifiers of all before journals opened in the system. This table is used after a system crash to determine which journals must be reopened and examined in order to perform a rollback operation. To preserve the integrity of this table, it is written out to disk automatically each time it is updated with a newly opened journal.

If a process opens the same before journal more than one time, the opening identifier received from the open_bj will be the same for each call. The process must close a before journal the same number of times it opens it, to render the journal inaccessible through the same opening identifier.

Entry: before_journal_manager_$set_default_bj

This entry point causes the specified before journal to become the default before journal. When no before journal is explicitly specified by the user at the beginning of a transaction, the default before journal for the process will be assigned to the transaction. The default before journal must be one of the before journals open in the process.

USAGE

```
declare before_journal_manager_$set_default_bj entry (bit(36) aligned,
      fixed bin(35));

call before_journal_manager_$set_default_bj (bj_opening_id, code);
```

ARGUMENTS

bj_opening_id
is the opening identifier of the before journal. (Input)

code
is a standard system error code. (Output)

NOTES

Several before_journal_manager_ entries expect an opening id to specify which before journal to use. If bj_opening_id is null, the following default assignments are attempted, in the order in which they are mentioned below, until one of them succeeds:

- The current default before journal in this process, if there is one; otherwise,
- The most recently open before journal among those that are still open, if there is one; otherwise,
- The system before journal. If the system before journal has not been opened yet in the current process, it is automatically opened.

Entry: before_journal_manager_$set_transaction_storage_limit

This entry point sets the maximum number of bytes a single transaction may use.

USAGE

```
declare before_journal_manager_$set_transaction_storage_limit entry
      (char (*), char (*), fixed bin (35), fixed bin (35));

call before_journal_manager_$set_transaction_storage_limit (dir_name,
      entryname, storage_limit, code);
```

ARGUMENTS

dir_name
    is the pathname of the containing directory. (Input)

entryname
    is the entryname of the before journal. (Input)

storage_limit
    is the maximum number of bytes a single transaction may use in the before journal. (Input)

code
    is a storage system status code. (Output)

Name: transaction_manager_

Entry points in transaction_manager_ begin and end transactions on behalf of users, return information about transactions, and recover transactions after system failure.

See the section entitled "Multics Data Management" in the *Multics Programmer's Reference Manual*, Order No. AG91, for a complete description of transactions and their use.

Entry: transaction_manager_$abandon_txn

This entry point relinquishes control of the current transaction, causing it to be adjusted (aborted unless a commit was already in progress) by the DM daemon (Data_Management.Daemon). The caller is immediately given a new TDT entry and can begin another transaction.

USAGE

```
declare transaction_manager_$abandon_txn entry (bit (36) aligned, fixed
      bin(35));

call transaction_manager_$abandon_txn (txn_id, code);
```

ARGUMENTS

txn_id
: is the identifier of the current transaction, or "0"b to default to the current transaction. (Input) If txn_id is neither "0"b nor the transaction identifier of the current transaction, dm_error_$transaction_not_current is returned. This argument can be used as a check to be sure which transaction is being abandoned.

code
: is a standard system status code. (Output) It can also be:

  dm_error_$no_current_transaction
  : No current transaction is defined for this process.

  dm_error_$not_own_transaction
  : A process can only abandon its own transaction.

  dm_error_$transaction_suspended
  : The current transaction is suspended and therefore cannot be abandoned.

Entry: transaction_manager_$abort_txn

This entry point aborts the current transaction, returning all modified DM files to the state they were in before the transaction began.

USAGE

```
declare transaction_manager_$abort_txn entry (bit(36) aligned, fixed
      bin(35));

call transaction_manager_$abort_txn (txn_id, code);
```

ARGUMENTS

txn_id
: is the identifier of the current transaction, or "0"b to default to the current transaction. (Input) If txn_id is neither "0"b nor the transaction identifier of the current transaction, dm_error_$transaction_not_current is returned. This argument can be used as a check to be sure which transaction is being aborted.

code
: is a standard system status code. (Output) It can also be:

  dm_error_$no_current_transaction
  : No current transaction is defined for this process.

  dm_error_$not_own_transaction
  : A process can only abort its own transaction.

  dm_error_$transaction_suspended
  : The current transaction is suspended and therefore cannot be aborted.

  dm_error_$unfinished_commit
  : The transaction was left in the middle of a commit operation. It is possible to call $commit_txn to complete the commit, or call either $abandon_txn or $kill_txn.

NOTES

If the transaction has already been abandoned, this entry point causes the DM daemon to abort it immediately.

This entry point will retry abort of a transaction that was left in an error state by a previous abort or rollback. It will not attempt abort of a transaction left in error by any other operation.

Entry: transaction_manager_$begin_txn

This entry point begins a transaction on behalf of the caller, by generating a unique transaction identifier and recording it in a TDT entry as the current transaction for the process. Other information, such as owner name, begin time, and transaction state (in-progress) are also recorded. The transaction id is passed to the before journal manager to begin the transaction.

USAGE

```
declare transaction_manager_$begin_txn (fixed bin(17), bit(36), bit(36)
      aligned, fixed bin(35));

call transaction_manager_$begin_txn (begin_mode,
      before_journal_opening_id, txn_id, code);
```

ARGUMENTS

begin_mode
: determines which of several protocols to use. (Input) The only mode currently available is normal mode.

  TM_NORMAL_MODE
  : requires locks to accompany all gets and puts, and requires all updates to be journalized.

before_journal_opening_id
: is the opening identifier of the before journal to be used by this transaction. (Input) If zero, a before journal is assigned by default to this transaction.

txn_id
: is the identifier of the newly created transaction. (Output) It is generated by transaction_manager_$begin_txn and is guaranteed to be unique across all Multics systems. Transaction identifiers are not reusable.

code
: is a standard system status code. (Output) It can also be:

  dm_error_$invalid_mode
  : The specified begin_mode is not currently supported.

  dm_error_$no_begins
  : Transactions are not allowed to be begun because DM daemon has disallowed beginning new transactions, for example when preparing to do a systemwide DMS shutdown.

  dm_error_$transaction_suspended
  : A transaction cannot be begun because a suspended one already exists.

  dm_error_$transaction_in_progress
  : A transaction cannot be begun because one is already active.

Entry: transaction_manager_$commit_txn

This entry point commits the current transaction. Any modifications made to DM files since the transaction began become permanent and visible to other transactions, as if all the changes were made in the same instant.

USAGE

```
declare transaction_manager_$commit_txn entry (bit(36) aligned, fixed
      bin(35));

call transaction_manager_$commit_txn (txn_id, code);
```

ARGUMENTS

txn_id
: is the identifier of the current transaction, or "0"b to default to the current transaction. (Input) If txn_id is neither "0"b nor the transaction identifier of the current transaction, dm_error_$transaction_not_current is returned. This argument can be used as a check to be sure which transaction is being committed.

code
: is a standard system status code. (Output) It can also be:

dm_error_$no_current_transaction
: No current transaction is defined for this process.

dm_error_$not_own_transaction
: A process can only commit its own transaction.

dm_error_$transaction_suspended
: The current transaction is suspended and therefore cannot be committed.

dm_error_$unfinished_abort
: The transaction was left in the middle of an abort operation. It is possible to call $abort_txn to complete the abort, or call either $abandon_txn or $kill_txn.

dm_error_$unfinished_rollback
: The transaction was left in the middle of a rollback operation. It is possible to call $rollback_txn to complete the rollback, call $abort_txn to abort the transaction, or call either $abandon_txn or $kill_txn.

NOTES

This entry point will retry commit of a transaction that was left in an error state by a previous commit. It will not, however, attempt to commit a transaction left in error by any other operation.

Entry: transaction_manager_$get_current_txn_id

This entry point returns the identifier of the current transaction, and tells whether the transaction is suspended or in error. See "Notes" below for a table of transaction identifiers and error codes returned.

USAGE

```
declare transaction_manager_$get_current_txn_id entry (bit(36) aligned,
      fixed bin(35));

call transaction_manager_$get_current_txn_id (txn_id, code);
```

ARGUMENTS

txn_id
is the identifier of the current transaction. (Output)

code
is one of the codes listed below. (Output)

NOTES

The txn_id and code values returned depend on the status of the current transaction:

	txn_id	code
1. Txn in progress.	valid id	0
2. No current txn.	0	dm_error_$no_current_transaction
3. Txn suspended.	valid id	dm_error_$transaction_suspended
4. Txn in error.	valid id	dm_error_$unfinished_abort
		or: dm_error_$unfinished_commit
		or: dm_error_$unfinished_rollback

Entry: transaction_manager_$get_txn_info

This entry point returns a structure containing all the information in the TDT about a transaction.

USAGE

```
declare transaction_manager_$get_txn_info entry (bit(36) aligned, ptr,
      fixed bin(35));

call transaction_manager_$get_txn_info (txn_id, txn_info_ptr, code);
```

ARGUMENTS

txn_id
is the identifier of a transaction, or "0"b to default to the current transaction. (Input)

txn_info_ptr
is a pointer to the txn_info structure, declared in dm_tm_txn_info.incl.pl1. (Input)

code
is a standard system status code. (Output)

ACCESS REQUIRED

The caller requires re access to dm_admin_gate_ to obtain information about another user's transaction.

STRUCTURE

This structure, declared in dm_tm_txn_info.incl.pl1, returns information about a transaction.

```
dcl 1 txn_info              aligned based (txn_info_ptr),
      2 version             char (8),
      2 txn_id              bit (36) aligned,
      2 txn_index           fixed bin,
      2 mode                fixed bin,
      2 state               fixed bin,
      2 error_code          fixed bin (35),
      2 checkpoint_id       fixed bin,
      2 rollback_count      fixed bin,
      2 owner_process_id    bit (36),
      2 owner_name          char (32),
      2 date_time_created   fixed bin (71),
      2 flags,
        3 (dead_process_sw,
           suspended_sw,
           error_sw,
           abandoned_sw,
           kill_sw)         bit (1) unaligned,
        3 mbz               bit (31) unaligned,
      2 journal_info        aligned,
        3 bj_uid            bit (36),
        3 bj_oid            bit (36),
        3 last_completed_operation
                            char (4),
        3 first_bj_rec_id   bit (36),
        3 last_bj_rec_id    bit (36),
        3 n_rec_written     fixed bin (35),
        3 n_bytes_written   fixed bin (35);
```

STRUCTURE ELEMENTS

version
is the version of the structure, currently TXN_INFO_VERSION_5.

txn_id
is the identifier of the transaction.

txn_index
is the index of the TDT entry for the transaction.

mode
is the begin_mode according to which the transaction was begun. See transaction_manager_$begin_txn for a list of modes.

state
is one of the states declared in the include file dm_tm_states.incl.pl1. It is either TM_IN_PROGRESS_STATE for an in-progress transaction, one of several intermediate states corresponding to calls made by the transaction manager (usually when the owner process has died in the middle of a call to transaction_manager_), or one of several error states corresponding to error codes returned by transaction_manager_.

error_code
is 0 or an error code returned by the last call made by the transaction manager.

checkpoint_id
is the identifier of the checkpoint that has most recently been rolled back to, or 0 for the start of the transaction.

rollback_count
is the number of times that the transaction has been rolled back, either by a rollback operation or as part of an unfinished abort.

owner_process_id
is the identifier of the process that began the transaction. This process may or may not still be running.

owner_name
is the Person.Project identifier of the process that began the transaction.

date_time_created
is the date-time that the transaction was begun.

dead_process_sw
is "1"b if the process that began the transaction is no longer running.

suspended_sw
is "1"b if the transaction is currently suspended.

error_sw
is "1"b if the transaction manager received an error code from one of its calls

(error_code ^= 0) and the transaction has not been adjusted since.

abandoned_sw
is "1"b if the transaction was abandoned by the owner via a call to $abandon_txn.

kill_sw
is "1"b if the owner called $kill_txn and the transaction is therefore waiting to be killed.

bj_uid
is the UID of the before journal chosen when the transaction was begun.

bj_oid
is the per-process opening identifier of the before journal used by the transaction.

last_completed_operation
is the name of the last completed before journal operation.

first_bj_rec_id
is the identifier of the first mark for this transaction.

last_bj_rec_id
is the identifier of the last mark for this transaction.

n_rec_written
is the number of marks that were written for this transaction.

n_bytes_written
is the total number of bytes written to the journal.

Entry: transaction_manager_$kill_txn

This entry point is intended to be called by the owner of a transaction when the owner cannot end the transaction normally and does not want the daemon to try to abort it for reasons of efficiency. Killing a transaction can destroy the consistency of the databases changed during the transaction, and is therefore appropriate only if consistency is no longer an issue (for example, if the databases are to be deleted). As with $abandon_txn, calling this entry point frees the user to begin a new transaction.

USAGE

```
declare transaction_manager_$kill_txn entry (bit(36) aligned, fixed
      bin(35));

call transaction_manager_$kill_txn (txn_id, code);
```

ARGUMENTS

txn_id
is the identifier of the transaction to be killed. (Input) If it is "0"b, the current transaction is used.

code
is a standard system status code. (Output) It can also be:

dm_error_$no_current_transaction
With txn_id="0"b, no current transaction is defined for this process.

dm_error_$transaction_suspended
With txn_id="0"b, the current transaction is suspended and therefore cannot be killed.

ACCESS REQUIRED

The caller requires re access to dm_admin_gate_.

Entry: transaction_manager_$resume_txn

This entry point reactivates a suspended transaction, once again allowing data operations on protected files.

USAGE

```
declare transaction_manager_$resume_txn entry (fixed bin(35));

call transaction_manager_$resume_txn (code);
```

ARGUMENTS

code
is a standard system status code. (Output) It can also be:

dm_error_$no_current_transaction
No current transaction is defined for this process.

dm_error_$no_suspended_transaction
The current transaction is not suspended.

Entry: transaction_manager_$rollback_txn

This entry point rolls the current transaction back to its beginning, by replacing all modifications to protected files caused by the transaction, with the before images preserved in the appropriate before journal. The transaction remains current for the user process.

USAGE

```
declare transaction_manager_$rollback_txn entry (bit(36) aligned, fixed
      bin, fixed bin(35));

call transaction_manager_$rollback_txn (txn_id, checkpoint_number,
      code);
```

ARGUMENTS

txn_id
: is the identifier of the current transaction, or "0"b to default to the current transaction. (Input) If txn_id is neither "0"b nor the transaction identifier of the current transaction, dm_error_$transaction_not_current is returned. This argument can be used as a check to be sure which transaction is being rolled back.

checkpoint_number
: must currently be 0. (Input)

code
: is a standard system status code. (Output) It can also be:

  dm_error_$no_current_transaction
  : No current transaction is defined for this process.

  dm_error_$not_own_transaction
  : A process can only roll back its own transaction.

  dm_error_$transaction_suspended
  : The current transaction is suspended and therefore cannot be rolled back.

  dm_error_$unfinished_abort
  : The transaction was left in the middle of an abort operation. It is possible to call $abort_txn to complete the abort, or call either $abandon_txn or $kill_txn.

  dm_error_$unfinished_commit
  : The transaction was left in the middle of a commit operation. It is possible to call $commit_txn to complete the commit, or call either $abandon_txn or $kill_txn.

NOTES

This entry point will retry rollback of a transaction that was left in an error state by a previous rollback. It will not attempt to rollback a transaction left in error by any other operation.

Entry: transaction_manager_$suspend_txn

This entry point puts the current transaction into a suspended state wherein it is temporarily unusable. Data operations to protected files are not allowed while the transaction is suspended, that is, until $resume_txn is called. Since the suspended transaction has not been completed, no new transaction can be begun.

USAGE

```
declare transaction_manager_$suspend_txn entry (fixed bin(35));

call transaction_manager_$suspend_txn (code);
```

ARGUMENTS

code
is a standard system status code. (Output) It can also be:

dm_error_$no_current_transaction
No current transaction is defined for this process.

dm_error_$transactions_suspended
The current transaction is already suspended.

NOTES

Suspension has the following effects:

1. The current transaction is temporarily unusable. As a result, the entry point $get_current_txn_id returns "0"b and the error code dm_error_$transaction_suspended.

2. No data operations on protected files are allowed while the transaction is suspended.

3. Both $begin_txn and $commit_txn return dm_error_$transaction_suspended.

4. Both $abort_txn and $adjust_tdt_entry (called by DMS) work on suspended transactions.

APPENDIX A

ERROR TABLES

The error codes used by MRDS are contained in a separate error table named mrds_error_. This error table is utilized in the same way as error_table_ (see "Handling Unusual Occurrences" in the Reference Manual) and contains those messages and error codes applicable to MRDS.

async_include_file_change
Include files no longer match.

attr_already_exists
The given attribute name has a previous definition.

attr_error
No attribute specification found following an attribute keyword.

bad_access_mode
Data base access mode is not a composite of r, s, m, d, or n.

bad_arith_const
An invalid arithmetic constant or value has been detected.

bad_attr
An illegal tuple attribute has been specified in the selection expression.

bad_attr_name
Attribute name contains an invalid attribute name character.

bad_builtin_obj
Unable to reference the scalar built-in functions.

bad_delim
A delimiter has been incorrectly specified.

bad_domain_proc
Encode/decode procedure could not be accessed.

bad_eq
An equal sign has been incorrectly specified.

bad_ident
An identifier contains invalid characters.

bad_invert_file_type
Entry is not a multisegment file.

bad_key_retrieve
Retrieval based on a primary key found multiple tuples.

bad_keyword
An expected keyword was not found.

bad_model
A file which is not a data model or is inconsistent has been specified.

bad_op
An arithmetic operator has been improperly specified in the -where clause.

bad_pathname
The pathname supplied is a control argument.

bad_quant
No tuple variable was specified following a quantifier.

bad_rel_name
Relation name contains an invalid relation name character.

bad_select_value
An unsupported data type was specified for a select item value.

bad_source_path
Source pathname is a control argument.

bad_temp_rel_val
A value specified for a temporary relation index is not an integer.

bad_var
An illegal tuple variable has been specified in the selection expression.

block_sel_incons
The number of items being selected is inconsistent among select blocks.

bool_leaf
An 'and' or 'or' operator has a constant or tuple attribute operand.

cant_ref_fun
Unable to reference a declared or built-in function.

comp_sel_expr
Complex selection expressions are not allowed for update operations.

conversion_condition
The conversion condition has been signalled during a data conversion attempt.

ctl_ent_is_dir
The control file path is a directory, not a vfile msf.

curr_not_alld
A -current operation is not permitted for a selection expression containing set operations.

db_already_open
Attempt to open a data base before previous openings have been closed.

db_busy
The specified data base is currently busy -- try later.

db_conflict_dead_process
A scope request cannot be honored due to a conflict with a nonpassive dead process.

diff_comp_domain
Attempt to compare attributes which are not defined over the same domain.

dom_integ
A value to be inserted into the data base does not satisfy integrity constraints.

domain_already_defined
The given domain name has a previous definition.

dup_invert_dir_name
     Inversion entry not a directory.

dup_not_alld
     A -dup is not allowed in a -current clause or in an operation other than retrieve.

dup_rel
     The given relation name has a previous definition.

dup_store
     A tuple with the specified primary key already exists.

dup_temp_rel_attr
     A non-unique attribute name was found in the definition of a temporary relation.

duplicate_key
     A tuple with the specified primary key already exists.

duplicate_opt
     A control option was given more than once.

duplicate_scope
     Attempt to define scope upon a file more than once.

empty_range
     No range definitions were found following a -range keyword.

empty_select
     No tuple attribute or tuple variable was specified following a -select or -current keyword.

empty_where
     No predicate follows the -where keyword.

error_condition
     The error condition has been signalled during a data conversion attempt.

expr_stack_ovfl
     Translator error -- expression stack overflow.

expr_syntax
     A syntax error has been detected within an arithmetic expression.

expression_not_complete
     A relation definition expression is not complete.

ext_data
     Data follows the right parenthesis.

fixedoverflow_condition
     The fixed overflow condition has been signalled during a data conversion attempt.

free_not_quiesced
     Attempt to free a data base which was not quiesced.

fun_syntax
     A syntax error has been detected within a function reference.

hold_quiesced_db
     Attempt to quiesce a data base before previously quiesced data bases have been freed.

ill_term
     There is an illegal term in the -where clause.

inv_comparison
    The data types cannot be compared.

inv_keyword
    An unrecognizable keyword was found in the selection expression.

This page intentionally left blank.

illegal_procedure_condition
     The illegal procedure condition has been signalled during a data conversion attempt.

inc_attr_acc
     Incorrect access to attribute.

inc_ready_mode
     The specified operation is not compatible with the current file ready mode.

inc_rel_acc
     Incorrect access to relation.

inc_secure_open
     Attempt to open secured data base from model, or through non-secure submodel.

incomp_se
     A selection expression of -another is valid only for a retrieve operation.

incomp_se_and_scope
     The selection expression was -another, but the scope has been changed from non-shared to shared mode.

incomplete_declaration
     Incomplete declaration.

incons_db
     There is an inconsistency in the data base. If this error persists, contact your Data Base Administrator.

inconsistent_close
     The data base has been closed -- but has been locked because of an inconsistency.

inconsistent_data_length
     The selection exression was -another, but the current data length is different than the previous call to retrieve.

inconsistent_database
     There is an inconsistency in the data base. If this error persists, contact your Data Base Administrator.

inconsistent_info
     An internal inconsistency has been detected.

inconsistent_num_files
     Number of files in data base does not match number specified in db_model.

inconsistent_options
     Options supplied cannot be used together.

inconsistent_submodel
     Inconsistent submodel.

inconsistent_transaction_se
     The selection expression was -another, but the original selection expression was in another transaction.

incorrect_dsmd_seq
     Data submodel definition entry called in incorrect sequence.

insuff_args
     There is no argument corresponding to a .V. in the selection expression.

internal_error
     Internal MRDS programming error. Please contact the MRDS developers.

inv_attr_name_first_char
     Invalid attribute name; attribute names must begin with an alphabetic character.

inv_literal_type
The value of a constant is not a string or arithmetic data type.

inv_operator
The relational operator index is not valid.

inv_rel_name_first_char
Invalid relation name; relation names must begin with an alphabetic character.

inv_string
An invalid string constant has been specified in the selection expression.

inv_string_len
An invalid repetition factor has been specified for a string constant.

inv_token
An unrecognizable token was found in the selection expression.

inval_del_expr
Invalid selection expression for delete.

inval_dtr_expr
Invalid selection expression for define_temp_rel.

inval_mod_expr
Invalid selection expression for modify.

inval_rtrv_expr
Invalid selection expression for retrieve.

invalid_db_index
Specified data base index does not correspond to currently open data base.

invalid_dec_data
Invalid data.

invalid_dm_descriptor
Data type given by descriptor not supported by Data Base Manager.

invalid_opening_mode
Invalid opening mode.

invalid_precision
Invalid precision specification.

invalid_rel
Submodel relation failed to perfectly validate against the model relation.

invalid_rel_index
An invalid relation index has been given.

invalid_scale
Invalid scale specification.

invalid_select_sets
An invalid select_sets sequence has been detected.

invalid_string_length
Invalid string length.

key_encd_ovfl
An overflow has occurred while encoding a floating point key/index value.

list_duplicate
A duplicate appears in the given list.

lit_string_ovfl
Translator error -- the literal area has overflowed.

long_ident
An identifier consists of more than 32 characters.

long_index
An index attribute is longer than the maximum key length allowed.

long_key
The primary key is longer than the maximum length allowed.

max_and_groups
Translator error -- maximum number of 'and' groups exceeded.

max_and_terms
Translator error -- maximum number of terms in 'and' group exceeded.

max_attributes
The maximum number of attributes allowed per relation has been exceeded.

max_expr_items
Too many items have been specified in an arithmetic expression.

max_indexes
The maximum number of secondaryily indexed attributes for a single relation has been exceeded.

max_rels
The maximum number of relation allowed per data base has been exceeded.

max_retr_len
The selected attributes exceeded the maximum temporary space available to hold them.

max_select_items
Too many items have been specified for selection in a -current or -select clause.

max_sf_args
The maximum number of scalar function arguments allowed has been exceeded.

max_temp_rels
The maximum number of temporary relation definitions has been exceeded.

max_tup_var
Too many tuple variables have been specified.

max_vars_rel
More tuple variables than iocb's for a given relation.

*

missing_relation_name
Relation name not specified.

missing_select
An expected -select clause was not found.

mixed_versions
Attempt to use different version data bases in same argument list.

*

mod_key_attr
Attempt to modify a key attribute.

mult_asts
Multiple asterisks followed an attribute name.

mult_att_def
An attribute has been multiply specified within a relation expression.

mult_att_ref
An attribute has been multiply referenced within a relation expression.

mult_def_var
A tuple variable has been multiply defined in the range clause.

mult_expr_vars
An arithmetic expression involving more than one tuple variable has been specified.

mult_index
A relation has been specified more than once in the index clause.

mult_paren
Multiple left parentheses were found.

multiple_tuples_found
A selection expression for modify resulted in more than one tuple being selected.

my_quiesced_db
Attempt to quiesce a data base which has already been quiesced by this process.

no_attr_lp
No attribute name was found following the left parenthesis.

no_attr_spec
None of the submodel attributes were found in the data model.

no_ctl_path
No control file path name was supplied.

no_current_tuple
No tuple was found which satisfied the selection expression.

no_database
No MRDS data base model found with the given pathname.

no_db_path
No data base path was supplied.

no_dms
Data management software could not be found.

no_domains
No domain specification found following a domain keyword.

no_dups_for_set_oper
Duplicates are not allowed in set operations.

no_inds
No index specification found following an index keyword.

no_key_specified
No key attribute field defined.

no_left_paren
No left parenthesis was found following the relation name.

no_model_access
Insufficient access to read data base model or submodel.

no_model_attr
The specified data model attribute name does not exist.

no_model_dom
The specified data model domain name does not exist.

no_model_rel
The specified data model relation name does not exist.

no_model_submodel
No data base model or submodel found with the given pathname.

*
no_prev_store
A -another keyword has been specified for store without a previous store.

no_primary_key
No primary key attributes were specified for the relation.

no_prior_se
A -another or -current keyword has been specified without a prior valid selection expression.

no_recursion
This command/subroutine may not be called recursively.

no_rel_attr
No attributes were specified for the relation.

no_rel_name
No relation name was found.

no_rels
No relation specification found following a relation keyword.

*
no_sel_exp
No selection expression was found.

no_sm_rel
No relation by this name exists in the submodel.

no_temp_dir
No temporary directory path was supplied.

no_tr_keys
No primary keys were designated in the selection expression.

no_tuple
There is no tuple satisfying the qualifications.

no_tuple_effect
Some of the tuple variables have no effect on the select set.

no_wakeup_user
A waiting and blocked data base user could not be awakened.

node_stack_ovfl
Translator error -- the node stack has overflowed.

non_scope_ready
File was not readied for scope_update or scope_retrieve.

not_dsm
The specified view pathname is not a data submodel.

not_freeing_area
The supplied area does not have the freeing attribute.

not_leaf
A 'not' operator has a constant or tuple attribute operand.

one_tuple_op
More than one tuple variable was selected for a modify or delete.

op_stack_ovfl
     Translator error -- the operator stack has overflowed.

open_name_already_known
     The open name given is already defined, open names must be unique.

open_name_not_known
     The given open name is not currently defined.

open_order
     There was an attempt to open an old version data base with new version data bases open.

overflow_condition
     The overflow condition has been signalled during a data conversion attempt.

parse_error
     Syntax error.

previously_defined_index
     An attribute was previously defined as an index.

process_not_found
     Unable to locate specified process in the data base control segment.

quiesce_pending
     Another process is waiting to quiesce the data base.

quiesce_too_few
     The number of data bases to quiesce is negative or zero.

quiesced_db
     The data base has been quiesced by another process.

quiesced_dead_db
     The data base has been quiesced by a process which no longer exists.

range_syntax
     A syntax error has been detected within a -range clause.

recursion_error
     This command/subroutine may not be called recursively.

rel_name_too_long
     The relation name exceeds the 30-character limit.

rel_node
     A relational operator has a term or group of terms as an operand.

rst_bad_attribute_count
     Model structure and attribute count don't agree.

rst_bad_bit_string
     Bit string violates syntax rules.

rst_bad_child_count
     Model structure and child link count don't agree.

rst_bad_declaration
     Error in the declaration of a domain.

rst_bad_domain_count
     Model structure and domain count don't agree.

rst_bad_encoding
     Source character was incorrectly encoded.

rst_bad_file_count
     Model structure and file count don't agree.

rst_bad_link_count
     Model structure and link count don't agree.

rst_bad_model
     Inconsistent data base model detected.

rst_bad_number_syntax
     Syntax error was found in a number.

rst_bad_relation_count
     Model structure and relation count don't agree.

rst_bad_semantics
     The intended meaning of a statement may be lost or misinterpreted.

rst_childless_parent
     The given foreign key has no child links.

rst_comment_ends_source
     Source segment ends in the middle of a comment.

rst_conversion_error
     Overflow occured trying to convert number to binary.

rst_dup_file
     The given file name has a previous definition.

rst_illegal_char
     Illegal character being skipped.

rst_inconsis_option
     The given attributes in a declaration are contradictory.

rst_invalid_structure_type
     The given number has no defined structure correspondence.

rst_io_error
     An error was detected during an I/O operation.

rst_link_attr_differ
     The parent/child attribute counts differ.

rst_list_delete_fail
     The item to be deleted was not in the list.

rst_list_duplicate
     Attempt to add a duplicate to the given list.

rst_logic_error
     Internal MRDS programming error. Please contact the MRDS developers.

rst_missing_file_model
     File model segment not found.

rst_missing_pathname
     An expected pathname was not found.

rst_missing_ref_domain
     A domain referenced by an attribute wasn't found.

rst_model_limit
     The capacity of the data base model has been exceeded.

rst_name_duplicate
     A relation's attribute list contains a duplicate name.

rst_name_too_long
A name exceeds it's maximum allowable length.

rst_no_key_attr
The given relation does not specify any key attributes.

rst_no_link_relation
The given link does not have a relation attached.

rst_not_rel_attr
A relation does not contain the referenced attribute.

rst_option_limit
The upper limit for an option's value was exceeded.

rst_parse_err_correct
Unable to understand statement structure, attempting guess at intended syntax.

rst_parse_err_no_correct
Unable to understand statment structure, and attempt at guessing intended syntax failed.

rst_parse_err_no_recover
Unable to comprehend statement structure, and attempt to recover by skipping to next recognizable delimiter failed.

rst_parse_err_recover
Unable to comprehend statement structure, skipping to next recognizable delimiter.

rst_parse_fail
Totally confused by statement syntax, unable to continue parsing.

rst_pathname_ends_source
The source segment ends during a path/entry name.

rst_rel_has_file
A referenced relation has a previous file definition.

rst_reserved_name
A reserved name was used.

rst_string_ends_source
The source segment ends within a quoted string.

rst_token_too_long
A token exceeds the maximum string size.

rst_undef_rel
A referenced relation has not been previously defined.

rst_undone_option
The specified option is not implemented.

rst_unused_attr
The given attribute has never been referenced in a data base relation.

rst_unused_attr_dom
The given domain has never been referenced in a data base relation.

rst_wrong_command
The command or subroutine call was given in an incompatible situation or sequence.

scal_func_conversion
A conversion condition was raised while processing a scalar function.

scope_empty
Attempt to delete scope tuple from empty scope set.

scope_mrds_access_conflict
The requested scope exceeds the MRDS access granted for this relation.

scope_not_empty
Attempt to define scope while scope is not empty.

scope_not_found
Specified scope tuple not in current scope.

scope_not_set
No scope currently set for the specified relation.

scope_system_access_conflict
The requested scope exceeds the system acl's on the given relation.

scope_viol
This operation is not permitted within the current scope definition.

sel_blk_synt
A syntax error has been detected within a select block.

select_mismatch
There are not enough value arguments to satisfy all specified select items.

select_syntax
A syntax error has been detected within a -select or -current clause.

sell_syntax
A syntax error has been detected within the selection expression.

set_ovfl
Too many select blocks have been specified in the selection expression.

set_syntax
Select blocks have been incorrectly combined.

size_condition
The size condition has been signalled during a data conversion attempt.

surplus_text
Text follows the logical end of the source segment.

too_many_args
The maximum number of expected arguments has been exceeded.

too_many_attributes
The maximum number of attributes for a relation has been exceeded.

too_many_data_models
Attempt to open more than the maximum number for data model openings.

too_many_dbs
Attempt to open more than the maximum allowable number of openings at one time.

too_many_open_names
Too many open names have been defined, some must be deleted first.

too_many_temp_files
The maximum number of temporary files has been exceeded.

trouble_lock
The data base is locked and may be inconsistent.

tuple_not_found
No tuple was found which satisfied the selection expression.

unable_to_create_channel
An event channel needed to activate a queued process could not be created.

unable_to_queue_user
A user could not be placed in the waiting queue due to an error.

unaccep_fn_args
A function reference includes an unacceptable argument, or the wrong number of arguments.

unbal_parens
The number of right parentheses does not match the number of left parentheses.

undef_attr
A referenced attribute has not been previously defined.

undef_fun
A referenced function is not built-in nor has it been declared.

undef_rel
A specified relation name is undefined in the submodel.

undef_temp_rel
The given index does not refer to a currently defined temporary relation.

undef_var
A specified tuple variable has not been previously defined.

undefined_domain
A referenced domain has not been previously defined.

undefined_temp_rel_index
The given index does not refer to a currently defined temporary relation.

underflow_condition
The underflow condition has been signalled during a data conversion attempt.

unknown_cursor_storage
The pointer to the storage for the cursor pointers is bad.

unknown_file_name
Specified relation name not known to this process.

unknown_proc_id
An unidentifiable data base user process has been encountered.

unknown_relation_name
Relation name specified is not in the current view of the data base.

unshared_opening
This operation is not valid for nonshared openings.

unsup_type
An unsupported data type has been specified as a value.

upd_temp_rel
Update operations are not permitted for temporary relations.

update_not_allowed
A relation is not available for update operations.

user_not_found
Unable to locate specified user in the data base control block.

var_stack_ovfl
Translator error -- the variable stack has overflowed.

version_1_dsm
Version 1 submodels are no longer supported by MRDS.

version_3_db
     Version 3 data bases are no longer supported by MRDS.

version_not_supported
     The data base is a version not supported by this command/subroutine.

view_prevent
     The specified operation cannot be accomplished using the current data base view.

where_syntax
     A syntax error has been detected within the -where clause.

*

APPENDIX B

MRDS DATA

Data that is specific to MRDS is contained in a table named mrds_data_. It provides changeable limits on the operation of MRDS.

Listed below are the parameters used during the compilation of some of the MRDS modules.

Data Item Name and Declaration	Value	Description
caller_define_temp_rel fixed bin(35)	4	translate called by define_temp_rel
caller_delete fixed bin(35)	1	translate called by delete
caller_modify fixed bin(35)	2	translate called by modify
caller_retrieve fixed bin(35)	3	translate called by retrieve
control_segment_name char(32)	db.control	name of data base concurrency control segment
current_version fixed bin(35)	4	current data base version
current_version_status fixed bin(35)	8	current version_status structure major number
dmd_version fixed bin(35)	4	version of model header structure
dsmd_version_number fixed bin(35)	5	version of submodel header structure
file_id_len_pad fixed bin(35)	7	length of file_id in bits in tuple_id
key_search_threshold fixed bin(35)	50	number of tuples selected before an additional key search, rather than comparisons against the selected set will be done.
lit_string_size fixed bin(35)	73728	max length of a literal string
lock_wait fixed bin(35)	900	wait time to lock acs control segment
lock_wait_time fixed bin(35)	30	set_scope default wait time
max_and_groups fixed bin(35)	100	max "and_groups" allowed in s.e. pred tree
max_and_terms fixed bin(35)	20	max terms allowed in an and_group in pred tree
max_attributes fixed bin(35)	256	max attrs allowed per relation by CMDB
max_builtin_args fixed bin(35)	4	max number of arguments to a built-in function
max_data_length fixed bin(35)	2000	max temp rel record data length
max_dbs fixed bin(35)	128	number of data base openings allowed
max_expr_items	20	stack depth for eval

```
   fixed bin(35)
max_expr_stack_size      14
   fixed bin(35)
max_id_len               32
   fixed bin(35)
max_kattr_len            253
   fixed bin(35)
max_key_len              253
   fixed bin(35)
max_line_size            50000
   fixed bin(35)
max_lit_string_size      254
   fixed bin(35)
max_pred_depth           30
   fixed bin(35)
max_pred_nodes           100
   fixed bin(35)
max_pred_ops             100
   fixed bin(35)
max_relations            256
   fixed bin(35)
max_select_items         100
   fixed bin(35)
max_sets                 20
   fixed bin(35)
max_sf_args              30
   fixed bin(35)
max_string_size          4096
   fixed bin(35)
max_td_len               10
   fixed bin(35)
max_temp_rels            20
   fixed bin(35)
max_token_size           65
   fixed bin(35)
max_tup_var              20
   fixed bin(35)
max_vfile_wait_time      60
   fixed bin(35)
normal_mode              1
   fixed bin
quiesce_mode             2
   fixed bin
quiesce_wait             900
   fixed bin(35)
statistics_update_count_interval
   fixed bin(35)         10
statistics_update_time_interval
   fixed bin(71)         300000000
statistics_update_small_rel_size
   fixed bin(35)         100
submodel_dir_name        secure.submodels
   char(16)
temp_seg_name            mrds_search_tidtemp.dbi
   char(23)
valid_id_chars           abcdefghijklmnopqrstuvwx
   char(128)             yzABCDEFGHIJKLMNOPQRSTUV
                         WXYZ0123456789_-
```

```
   of s.e. expressions
stack depth for eval
   of s.e. expressions
max character length of a
   tuple variable name
max length for key value

max total chars from attrs
   making up key field in rels
largest output line for
   cmdb listing
max repeated string
   literal size
size of stack for conversion
   pred tree to disj. norm. form
max number of pred tree
   tuple attr leaf nodes
max number of pred tree
   operator leaf nodes
largest number of relations
   cmdb can create
s.e. select clause max
   item count
s.e. max number of set
   operators
max number of args for
   scalar function
largest parsable token
   for cmdb
largest array space for
   token data
most simultaneous temp rels

largest s.e. token length

most s.e. tuple variables
   allowed
max time to wait for file
   operations for -share option
normal data base access mode

quiesce data base access mode

wait time to quiesce files

+number of rel ref times
   before statistics are next update
+real time til statistics
   next updated
+max size of rel to be
   updated every S.E.
name of submodel_dir in
   new architecture
common name for tid
   search temp segs
legal s.e. token characters
```

APPENDIX C

BIBLIOGRAPHY

Astrahan, N. M., et al, "System R: Relational Approach to Data Base Management," ACM Transactions on Data Base Systems, Vol. 1, No. 2. June 1976, pp 97-137

Chamberlain, D. D. and Boyce, P. F., "Sequel: A Structured English Query Language," Proc. ACM-SIGMOD Workshop on Data Description, Access, and Control, May 1974, ACM, New York 1974, pp 249-264
May 1974, ACM, New York 1974, pp 249-264

Codd, E. F., "A Relational Model of Data for Large Shared Data Banks," Comm. ACM 13, No. 6, June 1970 pp 377-387

Codd, E. F., "A Data Base Sublanguage Founded on the Relational Calculus," Proc. 1971 ACM-SIGFIDET Workshop

Codd, E. F., "Further Normalization of the Data Base Relational Model," Courant Computer Science Symposia 6 "Data Base Systems," New York City May 24-25 1971 Prentice Hall

Codd, E. F., "Normalized Data Base Structure: A Brief Tutorial," Proc. 1971 ACM-SIGFIDEG Workshop

Codd, E. F., "Relational Completeness of Data Base Sublanguages," Courant Computer Science Symposia 6, "Data Base Systems," New York City May 24-25, 1971 Prentice Hall

Date, C. J., "An Introduction to Data Base Systems," 2nd Edition, Reading, Mass. Addison Wesley, 1977

Sibley (Ed), E. H., "Special Issue: Data Base Management Systems," ACM Computing Surveys, Vol. 8, No. 1 March 1976

Won Kim, "Relational Data Base Systems," ACM Computing Surveys, Vol. 11, No. 3, Sept. 1979, pp 185-211

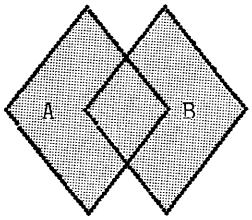
APPENDIX D

SET OPERATORS

Set operators are those operators used by MRDS which define the construction of different classes of selection expressions and which are based on the mathematical set theory operations of union, intersection, and difference. The three operations (including Venn diagrams) are defined as:

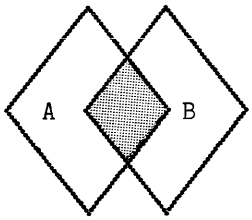
UNION

The union of A and B is defined to be the class of all the elements that belong either to A, or to B, or to both A and B.



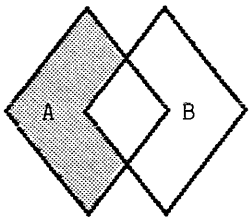
INTERSECTION

The intersection of A and B is defined to be the class of all the elements that belong to both A and B.



DIFFERENCE

The difference between A and B is defined to be the class of all elements that belong to A but do not belong to B.



APPENDIX E

ADMINISTRATOR-WRITTEN PROCEDURES

MRDS provides an interface to allow a DBA to write encoding, decoding, and check procedures for domains in a data base. Such procedures are associated with a domain at the time of data base creation and are executed when data defined by the domain is accessed. Additionally, the DBA may specify an internal data representation that is different from the way the data is represented to the external user. This feature may be used independently or in conjunction with the administrator-written procedures.

An encoding procedure, which is associated with a domain by using the -encode_proc domain option in the data base model source, is used to convert or translate external data input into a different data type or internal representation. For example, an encoding procedure may convert an alphabetic input item into all uppercase characters before it is stored in the data base. The encoding procedure is executed at two different times: when an attribute associated with the domain is stored or modified, and when an attribute associated with the domain is used in some types of selection expressions in such a way that a constant must be encoded during selection.

A decoding procedure, associated with a domain by the -decode_proc option, does the reverse of an encoding procedure. It converts or translates an internal data base value into its external representation. This procedure is also executed at two different times: when an attribute is retrieved from the data base and when an attribute associated with the domain is used in a selection expression in such a way that the data base value must be decoded into its external representation during selection. (Refer to the following paragraphs for additional information.)

Encoding and decoding procedures must not change the ordering of an item or selections will not work as expected (i.e., "abc" and "cba" should not be transformed to "cba" and "abc", respectively).

Qualifiers in a selection expression are compared without the use of encode or decode procedures; that is, all comparisons are done using the internal encoded data, except in two special cases.

1. When an attribute, that has an encode procedure associated with it, is compared against a constant. Encoded values must be compared with encoded values. For this case, the constant is encoded using the attribute's encode procedure, and the encoded result is compared against the encoded data base value.

2. When an expression is compared with an attribute, a constant, or another expression. Decoded values must be compared with decoded values. For this case, the data base values are decoded using the decode procedures for each attribute found in the expression. The expression is then evaluated and the decoded result is compared as follows. If it is being compared against a constant, the constant value is used directly because it is already decoded. If it is being compared with an attribute, the attribute value is decoded using its decode procedure. If it is being compared against another expression, the expression is evaluated as just described and the decoded results are compared.

Check procedures, specified by the -check_proc domain option, are used to verify data validity prior to its storage in the data base. These procedures are called whenever data defined by the domain is stored or modified. The procedure does whatever checking is desired by the DBA and returns a true or false value depending on whether or not the data is acceptable. This is done immediately before the data is stored and after any encoding or conversion has been done. The data verified is already in its internal format.

The DBA may use the -decode_dcl domain option to specify data conversion from an external to an internal data type either with or without encoding or decoding. For example, to speed processing or save space a data item may have an external representation of character, but be stored as binary. Such simple conversion may be done by using the -decode_dcl option alone without any encoding/decoding procedures. If the -decode_dcl option is used with an encode or decode procedure, it defines the user-visible data type that is processed by that procedure.

CODING ADMINISTRATOR-WRITTEN PROCEDURES

To interface properly with MRDS, encoding, decoding, and check procedures must be written as recorded below. The example gives a sample data base and procedures, and describes in detail when the procedures are called.

Encoding Procedure

The encoding procedure is used to convert a data item to be stored or modified into a different internal representation. Once defined for a domain, the procedure may not be moved or deleted without causing store and some selection operations using this domain to fail. It can, however, be changed without re-creating the data base; this does not change existing values already stored in the data base. Encoding procedures receive three arguments, each accompanied by standard Multics descriptors. Administrators who wish to write encoding procedures to accept a variety of input and output data types may use these descriptors. See the MPM Subsystem Writers' Guide for more information about argument list format and descriptors.

Usage

```
encode_proc:  procedure (user_value, db_value, code);
```

where:

1. user_value (Input)
   is the value, in user-visible format, to be encoded. If the -decode_dcl domain option is used, this argument matches that declaration. Otherwise, it matches the internal domain declaration.

2. db_value (Output)
   is the value in the format suitable for storage in the data base. It must match the internal domain declaration.

3. code (Output)
   is a status code. A nonzero code returned by the encoding procedure terminates the data base operation in progress. The actual code, however, is discarded. Therefore, if the DBA wishes to issue explanatory messages about the error, the encoding procedure should do so using the sub_err_ subroutine documented in the MPM Subsystem Writers' Guide.

Notes

The encoding procedure may convert and reformat data for storage in the data base. It should not, however, change the relative ordering of data or selections may not work as expected.

If an encoding procedure and a decoding procedure are both used for a domain, they should perform symmetrical transformations on the data (i.e., if "abc" encodes to "ABC", then "ABC" should decode to "abc"). If this is not done, the result of data base retrievals is unpredictable.

Decoding Procedure

The decoding procedure is used to convert a data item stored in the data base into its external representation after retrieval for the user. Once defined for a domain, the procedure may not be moved or deleted without causing retrieve and some selection operations using this domain to fail. It can, however, be changed without re-creating the data base. Decoding procedures receive three arguments, each accompanied by standard Multics descriptors. Administrators who wish to write decoding procedures to accept a variety of input and output data types may use these descriptors. See the MPM Subsystem Writers' Guide for more information about argument list format and descriptors.

Usage

```
decode_proc:  procedure (db_value, user_value, code);
```

where:

1. db_value (Input)
   is the value as it appears in the data base. It matches the internal domain declaration.

2. user_value (Output)
   is the value in user-visible format. If the -decode_dcl domain option is used, this argument must match that declaration. Otherwise, it matches the internal domain declaration.

3. code (Output)
   is a status code. A nonzero code returned by the decoding procedure terminates the data base operation in progress. The actual code, however, is discarded. Therefore, if the DBA wishes to issue explanatory messages about the error, the decoding procedure should do so using the sub_err_ subroutine documented in the MPM Subsystem Writers' Guide.

Notes

If an encoding procedure and a decoding procedure are both used for a domain, they should perform symmetrical transformations on the data (i.e., if "abc" encodes to ""ABC", then "ABC" should decode to "abc"). If this is not done, the result of data base retrievals is unpredictable.

Check Procedure

The check procedure is used to ensure that data to be stored in the data base passes DBA-defined integrity tests. This procedure is called as a function with one argument (the value to be stored in the data base) and returns a true or false value depending on whether or not the value is acceptable. Once defined for a domain, the procedure may not be moved or deleted without causing store operations using this domain to fail. It may, however, be changed without re-creating the data base. The argument passed to this procedure is accompanied by a standard Multics data descriptor. Administrators who wish to write check procedures to accept a variety of data types may use this descriptor. See the MPM Subsystem Writers' Guide for more information about argument list format and descriptors.

Usage

```
check_proc:  procedure (user_value) returns (fixed bin(35));
```

where:

1. user_value (Input)
   is the value to be stored in the data base. It matches the internal domain declaration.

2. OK (Output) fixed bin(35)
   is an indicator that is:

   1 (true) if the value is acceptable
   0 (false) if it is not

Note

The check_proc is called after the encoding procedure, if any.

Example

1. For the data base defined by:

```
domain:  name char (32),
         birthdate fixed bin (71) /* internal representation */
            -decode_dcl char(17)          /* external representation */
            -encode_proc >udd>Proj>DBA>encode
            -decode_proc >udd>Proj>DBA>decode
            -check_proc >udd>Proj>DBA>check;

relation: birth_info (name* birthdate);
```

2. The encoding procedure is:

```
encode: procedure (user_value, db_value, code);
dcl user_value char(*);       /*external data type */
dcl db_value fixed bin(71);   /* internal data type */
dcl code fixed bin(35);       /* status code */
dcl convert_date_to_binary_ entry
    char(*), fixed bin(71), fixed bin(35));

code = 0;
call convert_date_to_binary_
     (user_value, db_value, code);
return;
end encode;
```

3. The decoding procedure is:

```
decode: procedure (db_value, user_value, code);
dcl db_value fixed bin(71);   /* internal data type */
dcl user_value char(*);       /* external data type */
dcl code fixed bin(35);       /* status code */
dcl date_time_ entry
    (fixed bin(71), char(*));

code = 0;
call date_time_ (db_value, user_value);
```

```
return;
end decode;
```

4. And the check procedure is:

```
check: procedure (user_value) returns (fixed bin(35));
dcl user_value fixed bin(71);          /* value to be stored after check */
dcl OK fixed bin(35);                  /* return indicator */
dcl clock entry
   returns (fixed bin(71));

if user_value < clock_ ()              /* compare with current time */
then OK = 1;                           /* only want times in past */
else OK = 0;                           /* future times are no good */
return (OK);
end check;
```

5. The interaction of these procedures for the following operations is described below (assuming that the data base has been opened with an index of 1).

```
call dsl_$store (1, "birth_info", "John Doe", "12/25/7 07:30", code);
```

Encode is called with user_value equal to "12/25/79 07:30" and returns its binary clock equivalent in db_value.

Check is called with that clock value and returns true since it is a date in the past.

The tuple is stored in the data base.

```
call dsl_$retrieve (1, "-range (info birth_info) -select info.name
   info.birthdate", name, birthdate, code);
```

Decode is called with db_value equal to the binary clock value stored for birthdate. It will return a user_value of "12/25/79 0730.0".

"John Doe" is returned in name and "12/25/79 0730.0" is returned in birthdate.

```
call dsl_$store (1, "birth_info", "Richard Roe", "May 1 1999 0849.", code);
```

Encode is called with user_value = "May 1 1999 0849." and returns its binary clock equivalent in db_value.

Check is called with that clock value and returns false since the date is in the future.

The error code mrds_error_$dom_integ is returned to the calling program and the tuple is not stored.

```
call dsl_$store (1, "birth_info", "Richard Roe", "May 1 1979 0849.", code);
```

Encode is called with user_value = "May 1 1979 0849." and returns its equivalent in db_value.

Check is called with the clock value and returns true.

The tuple is stored in the data base.

```
call dsl_$retrieve (1, "-range (info birth_info) -select info.name
   -where info.birthdate > ""10/01/79 0000."" ", name, code);
```

Encode is called with user_value equal to "10/01/79 0000." and returns its binary clock equivalent.

The data base is searched for a qualifying tuple. "John Doe" is returned in name.

```
call dsl_$retrieve (1, "-range (info birth_info) -select info.name
   -where [substr (info.birthdate, 1, 2)] = ""05""", name, code);
```

Decode is called to convert the binary clock value for each data base tuple into a character string that is input to the substr function.

"Richard Roe" is returned in name.

APPENDIX F

MRDS INCLUDE FILES

For dsl_ Entries:

mrds_attribute_list

NOTE: Some of the include files may reference version 3 data bases which are no longer supported. Please disregard these references.

Description:

For a given data base opening via a model or submodel view, this structure contains the following attribute information for a particular relation: the number of attributes in this model/submodel view of the relation and the names in both the model and submodel (these will be the same if opened with a model view), the name of the domain for each attribute, the descriptor of the user's view of the data type, and whether the attribute can be used as an indexed attribute.

Access information is also returned for various versions of MRDS security as follows:

system_acl entries refer strictly to "rew" type Multics ACLs.

mrds_access entries are version-dependent. Version 4 data bases released in MR8 have no MRDS-specific access, but use system ACLs of "rew". Version 4 data bases for MR9.0 MRDS using submodel security have MRDS specific access mode of append/delete_tuple for relations and read/modify_attr for attributes.

effective_access entries use the same units as mrds_access. This is the logical result of applying both MRDS and system access and coming up with a user-effective mode of access to the relation/attribute.

```
declare 1 mrds_attribute_list aligned based (mrds_attribute_list_ptr),
        2 version fixed bin,     /* version number of this structure */
        2 access_info_version fixed bin,
                                 /* version of MRDS access modes
                                    3 => version 3 db with r-s-m-d access,
                                    4 => version 4 MR8 db with r-e-w access,
                                    5 => version 4 MR9 db with relation a-d, and
                                    attr r-m modes (submodel security) */
        2 num_attrs_in_view fixed bin,
                                 /* number of attributes in this view of the
                                    relation */
        2 submodel_view bit (1) unal,
                                 /*  ON = > the opening was via a submodel */
```

```
        2 mbz1 bit (35) unal,
        2 attribute (mrds_attribute_list_num_attrs_init refer
            (mrds_attribute_list.num_attrs_in_view)),
          3 model_name char (32),     /* name of attribute in model */
          3 submodel_name char (64), /* alias name of attribute in submodel,
                                        else model name */
          3 domain_name char (32),    /* name of the domain for this attribute */
          3 user_data_type bit (36), /* standard Multics data descriptor for storage
                                        format user's view if -decode_dcl, else
                                        same as db descriptor */
          3 system_acl char (8) varying,
                                     /* the system access from r-e-w modes */
          3 mrds_access char (8) varying,
                                     /* version 3 => from r-s-m-d,
                                                 4 => from r-e-w,
                                                 5 => from r-w */
          3 effective_access char (8) varying,
                                     /* effect of system + MRDS access units */
          3 indexed bit (1) unal,    /* ON => this is a secondary index
                                        attribute, or a key head */
          3 mbz2 bit (35) unal ;

declare  mrds_attribute_list_num_attrs_init fixed bin ;

declare  mrds_attribute_list_ptr ptr ;

declare  mrds_attribute_list_structure_version fixed bin init (1) int static
           options (constant) ;
```

---

mrds_database_list

Description

This structure is used by mrds_dsl_list_dbs to return an array of data base opening information. The data bases which are opened for the calling process have their opening index and opening model or submodel pathname returned in the array.

```
declare  database_list_ptr ptr ;        /* points to array of indexes/pathnames */

declare 1 database_list aligned based (database_list_ptr),
                                        /* array of paths/indexes */
        2 number_open fixed bin,        /* total open by this process */
        2 db (number_of_openings refer (database_list.number_open)),
                                        /* array of open db info */
          3 index fixed bin (35),       /* data base opening index */
          3 path char (168) ;           /* model or submodel opening pathname */

declare  number_of_openings fixed bin ; /* total number open by this process */
```

---

mrds_database_openings

Description:

This structure is used by dsl_$list_openings to return an array of data base opening information. The mrds_databases which are opened for the calling process have their opening index and opening model or submodel pathname returned in the array.

```
declare 1 mrds_database_openings aligned based (mrds_database_openings_ptr),
                                 /* array of paths/indexes */
        2 version fixed bin,     /* the version number of this structure */
        2 number_open fixed bin, /* total open by this process */
        2 mbz1 bit (36) unal,
        2 db (mrds_database_openings_num_open_init
         refer (mrds_database_openings.number_open)),
                                 /*array of open db info */
          3 index fixed bin (35),    /* data base opening index */
          3 path char (168),         /* model or submodel opening pathname */
          3 mode char (20),          /* opening mode of the data base */
          3 model bit (1) unal,      /* on => opened via the model */
          3 submodel bit (1) unal,   /* on => opened via a submodel */
          3 mbz2 bit (34) unal ;

declare  mrds_database_openings_ptr ptr ;
                                 /* points to array of indexes/pathnames */

declare  mrds_database_openings_num_open_init fixed bin ;
                                 /* total number open by this process */

declare  mrds_database_openings_structure_version fixed bin init static
           options (constant) init (1) ;
                                 /* current version */
```

---

mrds_new_scope_modes

Description:

This include file defines named constants which can be used to specify the MRDS operations to be permitted and prevented in a call to dsl_$set_scope.

```
dcl (NO_OP init (0),
     READ_ATTR init (1),
     APPEND_TUPLE init (2),
     DELETE_TUPLE init (4),
     MODIFY_ATTR init (8),
     UPDATE_OPS init (14),
     ALL_OPS init (15)) fixed bin int static options (constant);
```

---

mrds_opening_modes_

Description:

This include file defines named constants which can be used in calls to dsl_$open when opening a MRDS data base.

```
dcl (RETRIEVAL              init(1),
UPDATE                      init(2),
EXCLUSIVE_RETRIEVAL         init(3),
EXCLUSIVE_UPDATE            init(4)) fixed bin(35) int static options(constant);
```

mrds_path_info

Description:

This structure returns information about a relative pathname. The information returned is the absolute pathname. In the case that the relative path points to a MRDS data base or submodel,it returns information defining whether it is a model or a submodel, the MRDS version of the model or submodel, its creator, and the time of creation.

```
declare 1 mrds_path_info aligned based (mrds_path_info_ptr),
        2 version fixed bin,          /* version number for this structure */
        2 absolute_path char (168), /* the absolute path from the input
                                       relative path */
        2 type,
          3 not_mrds bit (1) unal,  /* on => path not to model or submodel */
          3 model bit (1) unal,     /* on => path to data base model, thus
                                       possible .db suffix */
          3 submodel bit (1) unal,  /* on => path to submodel, thus possible .dsm
                                       suffix */
          3 mbz1 bit (33) unal,
        2 mrds_version fixed bin,   /* the mrds version number of the model or
                                       submodel */
        2 creator_id char (32),     /* the person.project.tag of the creator */
        2 creation_time fixed bin (71),
                                    /* convert date to binary form of time
                                       model/submodel created */
        2 mbz2 bit (36) unal ;

declare  mrds_path_info_ptr ptr ;

declare  mrds_path_info_structure_version fixed bin init (1) int static options
           (constant);
```

mrds_relation_list

Description:

For a given opening of a data base via a model or submodel view, this structure will contain the list of relations as seen from that view. It contains the number of relations in that view and both the submodel and model names of the relation (model = submodel name if not a submodel opening) as well as whether the opening was via a submodel or not. The virtual relation bit indicates when the model name may not be valid due to a mapping over more than one relation in the model.

Access information for various versions of MRDS access is also returned, as follows:

system_acl entries refer strictly to "rew" type Multics ACLs.

mrds_access entries are version-dependent. Version 4 data bases released in MR8 have no MRDS-specific access, but use system ACLs of "rew". Version 4 data bases for MR9.0 MRDS using submodel security have MRDS-specific access mode of append/delete_tuple for relations and read/modify_attr for attributes.

effective_access entries use the same units as mrds_access. This is the logical result of applying both MRDS and system access and coming up with a user-effective mode of access to the relation/attribute.

```
declare 1 mrds_relation_list aligned based (mrds_relation_list_ptr),
        2 version fixed bin,              /* version number for this structure */
        2 access_info_version fixed bin, /* version of MRDS access modes
                                            3 => version 3 db with r-s-m-d
                                                 access,
                                            4 => version 4 MR8 db with r-e-w
                                                 access
                                            5 => version 4 MR9 db with relation
                                                 a-d, and attr r (submodel
                                                 security) */
        2 num_rels_in_view fixed bin,     /* count of relations present in this
                                            view */
        2 submodel_view bit (1) unal,     /* ON => this opening was via a
                                            submodel */
        2 mbz1 bit (35) unal,
        2 relation (mrds_relation_list_num_rels_init refer
          (mrds_relation_list.num_rels_in_view)),
          3 model_name char (32),         /* name of relation in data base
                                            model */
          3 submodel_name char (64),      /* alias name of relation in submodel,
                                            else model name */
          3 system_acl char (8) varying, /* the system access from r-e-w modes */
          3 mrds_access char (8) varying, /* version 3 => from r-s-m-d,
                                                       4 => from r-e-w,
                                                       5 => from a-d */
          3 effective_access char (8) varying,
                                          /* effect of system + MRDS access,
                                             units */
          3 virtual_relation bit (1) unal,
                                          /* ON => submodel relation defined over
                                             >1 model relation */
          3 mbz2 bit (35) unal ;

declare  mrds_relation_list_num_rels_init fixed bin ;

declare  mrds_relation_list_ptr ptr ;

declare  mrds_relation_list_structure_version fixed bin init (1) int static
           options (constant) :
```

FOR mmi_ ENTRIES:

mrds_authorization

Description:

This structure returns the caller's user_class--either data base administrator or normal user. Note that these separate classes were used to allow future expansion to the user classes (rather than make them logical "not"'s of one another). NOTE: a DBA is always also a normal user. Thus if the caller is a DBA, his normal_user bit will also be on.

```
declare 1 mrds_authorization aligned based (mrds_authorization_ptr),
        2 version fixed bin,           /* version number of this structure */
        2 administrator bit (1) unal, /* caller is a DBA */
        2 normal_user bit (1) unal,   /* caller has no special privileges */
        2 mbz bit (34) unal ;

declare  mrds_authorization_ptr ptr ; /* pointer for referring to the
                                         structure */

declare  mrds_authorization_structure_version fixed bin init (1) int static
           options (constant) ;
```

mrds_database_state

Description:

This structure returns the data base state (secured or unsecured) for determining how commands and subroutines will behave for each case. The secured bit was kept separate from the unsecured, rather than its logical "not", to allow for future extensibility of data base secured states.

```
declare 1 database_state aligned based (database_state_ptr),
        2 version fixed bin,        /* version number of this structure */
        2 unsecured bit (1) unal,   /* data base not secured */
        2 secured bit (1) unal,     /* data base has been secured */
        2 mbz bit (34) unal ;

declare  database_state_ptr ptr ;  /* pointer for referring to the structure */

declare  database_state_structure_version fixed bin init (1) int static options
           (constant) ;
```

---

mrds_db_model_info

Description:

This structure passes back information common to the whole data base, rather than that pertaining to a particular relation or attribute. It refers to the data base model, rather than to some submodel for that model.

```
declare 1 mrds_db_model_info aligned based (mrds_db_model_info_ptr),
        2 version fixed bin,              /* version number for this structure */
        2 model_version fixed bin,        /* the version number of the data base
                                             model */
        2 creator_id char (32),           /* the person.project.tag of the data
                                             base creator */
        2 creation_time fixed bin (71),   /* the convert date to binary form of
                                             the data base creation time  */

        2 mbz bit (36) unal ;

declare  mrds_db_model_info_ptr ptr ;

declare  mrds_db_model_info_structure_version fixed bin int static options
           (constant) init (1) ;
```

---

mrds_db_model_relations

Description:

This structure returns the list of all relation names in the data base model. A count of the number of names present is included. No submodel alias names for the relations are involved.

```
declare 1 mrds_db_model_relations aligned based (mrds_db_model_relations_ptr),
        2 version fixed bin,          /* version number for this structure */
        2 relation_count fixed bin,   /* total number of relations in this model */
```

```
        2 mbz1 bit (36) unal,
        2 relation (mrds_db_model_relations_count_init refer
            (mrds_db_model_relations.relation_count)),
          3 name char (32),          /* name of the relation in the model */
          3 mbz2 bit (36) unal ;

declare  mrds_db_model_relations_ptr ptr ;

declare  mrds_db_model_relations_count_init fixed bin ;

declare  mrds_db_model_relations_structure_version fixed bin int static init
           (1) options (constant) ;
```

---

mrds_db_model_rel_attrs

Description:

This structure returns, for a given relation, the list of all attribute names in the data base model. A count of the number of names present is included. No submodel alias names for the attributes are involved. Also, the domain name and the user's view descriptor for the data type is returned, as well as a bit indicating whether the attribute can be used as if it were indexed or not.

```
declare 1 mrds_db_model_rel_attrs aligned based (mrds_db_model_rel_attrs_ptr),
        2 version fixed bin,          /* version number for this structure */
        2 attribute_count fixed bin,  /* total number of attributes in this
                                         model */
        2 mbz1 bit (36) unal,
        2 attribute (mrds_db_model_rel_attrs_count_init refer
            (mrds_db_model_rel_attrs.attribute_count)),
          3 name char (32),           /* name of the attribute in the model */
          3 domain char (32),         /* the name of the underlying domain for
                                         this attribute */
          3 user_data_type bit (36),  /* standard Multics descriptor for the
                                         user's view of the data storage
                                         layout */
          3 indexed bit (1) unal,     /* on => key head or secondarily indexed
                                         attribute */
          3 mbz2 bit (35) unal ;

declare  mrds_db_model_rel_attrs_ptr ptr ;

declare  mrds_db_model_rel_attrs_count_init fixed bin ;

declare  mrds_db_model_rel_attrs_structure_version fixed bin int static init (1)
           options (constant) ;
```

For msmi_ Entries:

mrds_dsm_attribute_data

Description:

This include file contains information about all the attributes in a relation.

```
dcl 01 mrds_dsm_attribute_data aligned based (mrds_dsm_attribute_data_ptr),
    02 version fixed bin,
    02 number_of_attributes fixed bin,
    02 attributes (mrds_dsm_attribute_data_num_atts
     refer (mrds_dsm_attribute_data.number_of_attributes)),
      03 submodel_attribute_name char (64),
      03 model_attribute_name char (32),
      03 read_access bit (1) unal,
      03 modify_access bit (1) unal,
      03 null_access bit (1) unal,
      03 mbz1 bit (33) unal;

dcl  mrds_dsm_attribute_data_ptr ptr;

dcl  mrds_dsm_attribute_data_num_atts fixed bin;

dcl  mrds_dsm_attribute_data_structure_version fixed bin init (1) internal
       static options (constant);
```

---

mrds_dsm_relation_data

Description:

This include file contains information about all the relations in a submodel view.

```
dcl 01 mrds_dsm_relation_data aligned based (mrds_dsm_relation_data_ptr),
    02 version fixed bin,
    02 number_of_relations fixed bin,
    02 relations (mrds_dsm_relation_data_num_rels
     refer (mrds_dsm_relation_data.number_of_relations)),
      03 submodel_relation_name char (64),
      03 model_relation_name char (32),
      03 append_access bit (1) unal,
      03 delete_access bit (1) unal,
      03 null_access bit (1) unal,
      03 mbz1 bit (33) unal;

dcl  mrds_dsm_relation_data_ptr ptr;

dcl  mrds_dsm_relation_data_num_rels fixed bin;

dcl  mrds_dsm_relation_data_structure_version fixed bin init (1) internal static
       options (constant);
```

---

mrds_dsm_submodel_info

Description:

This include file contains the structure returned by msmi_$get_submodel_info.

```
dcl 01 mrds_dsm_submodel_info based (mrds_dsm_submodel_info_ptr),
    02 version fixed bin,                  /* version of this structure */
    02 submodel_version fixed bin,         /* version of the submodel */
    02 database_path char (168),           /* absolute path of the data base that
                                              the submodel refers to */
    02 submodel_path char (168),           /* absolute path of the submodel (may
                                              be a link) */
    02 date_time_created fixed bin (71),   /* date-time submodel was created in
                                              standard format */
    02 creator_id char (32);               /* Person.Project.Tag of the submodel
                                              creator */

dcl  mrds_dsm_submodel_info_ptr ptr;       /* pointer to the structure */

dcl  mrds_dsm_submodel_info_structure_version fixed bin init (1) internal static
       options (constant);
```

For dmd_ Entries (obsolete):

mrds_dm_header

```
dcl 1 dm_header based (dmh_ptr),     /* data model header */
    2 dm_header_id char (8),         /* identification as data model header */
    2 dmd_version fixed bin,         /* version number of dmd_ creating this
                                        model */
    2 creator_id char (32),          /* group id of creator */
    2 create_time fixed bin (71);    /* time of creation */

dcl  dmh_ptr ptr;
```

---

mrds_model_relations

```
dcl 1 model_relations based (mr_ptr),      /* structure to return names of all
                                              relations in a model */
    2 nrels fixed bin (10),                /* number of relations */
    2 relation_name (num_relations_alloc refer (model_relations.nrels))
       char (32);                          /* relation names */

dcl  num_relations_alloc fixed bin (10);   /* number of relations in model for
                                              allocation purposes */

dcl  mr_ptr ptr;
```

---

mrds_rel_desc

```
dcl 1 rel_desc based (rd_ptr),             /* record description of relation
                                              records */
    2 num_attr fixed bin,                  /* number of attributes in the model */
    2 key_length fixed bin (35),           /* length in bits of data portion of
```

```
                                        tuple */
    2 data_length fixed bin (35),       /* length in bits of data portion of
                                           tuple */
    2 num_keys fixed bin,               /* number of key attributes */
    2 inversion bit (1) unal,           /* On if this relation contains any
                                           inverted attributes */
    2 reserved bit (35) unal,           /* Reserved for future use */
    2 attributes (num_attr_alloc refer (rel_desc.num_attr)),
      3 attribute_name char (32),       /* name of attribute  */
      3 domain_name char (32),          /* name of underlying domain */
      3 bit_offset bit (18) unaligned,  /* offset within tuple of data item */
      3 bit_length bit (18) unaligned,  /* length of data item in bits */
      3 key_flag bit (1) unaligned,     /* indicates whether attribute is part
                                           of primary key */
      3 inver_flag bit (1) unaligned,   /* On if this attribute is inverted */
      3 unused bit (34) unaligned,      /* reserved for expansion */
      3 key_attr_order fixed bin,       /* order num of this key attr */
      3 descriptor bit (36);            /* Multics descriptor for attribute */

dcl  num_attr_alloc fixed bin (10);     /* Number of attributes in relation for
                                           allocation purposes */

dcl  rd_ptr ptr;
```

For dsmd_ Entries (obsolete):

mrds_dsm_display_rels

```
dcl 1 dsm_display_rels based (drel_ptr), /* user-specified relations for
                                            display */
    2 nrels fixed bin,                  /* number of relations */
    2 relation (nrels_alloc refer (dsm_display_rels.nrels)) char (32);
                                        /* relation names */

dcl  nrels_alloc fixed bin;

dcl  drel_ptr ptr;
```

---

mrds_dsm_header_str

```
dcl 1 dsm_header_record based,                  /* Data submodel header str */
    2 dsm_generator_version fixed bin init (1), /* Generator version number */
    2 date_time_generated fixed bin (71),       /* Date time of generation */
    2 database_pn char (168),                   /* Data base pathname */
    2 name char (32),                           /* Header name */
    2 num_of_relations fixed bin (35),          /* Total number of relations
                                                   in this data submodel */
    2 creator_id char (32);                     /* The ID of the person
                                                   creating the submodel */
```

---

mrds_dsm_rel_str

```
dcl 1 dsm_relation_str based,           /* dsm relation structure */
    2 key,                              /* vfile_ key */
      3 submodel_rel_name char (32),    /* Submodel_relation name */
    2 record,                           /* vfile record */
      3 model_rel_name char (32),       /* Model relation name */
      3 no_attributess fixed bin,       /* Number of attributes in this
                                           relation */
      3 attribute_info (dsm_num_attr_alloc refer (no_attributes)),
        4 submodel_att_name char (32),  /* Submodel attribute name */
        4 model_att_name char (32);     /* Model attribute name */

dcl  dsm_num_attr_alloc fixed bin;      /* Number of attributes in relation for
                                           allocation purposes */
```

MULTICS RELATIONAL DATA STORE

REFERENCE MANUAL ADDENDUM B

SUBJECT

Changes to the Manual

SPECIAL INSTRUCTIONS

This is the second addendum to AW53, Revision 4, dated September 1981. Refer to the Preface for "Significant Changes."

Insert the attached pages into the manual according to the collating instructions on the back of this cover. Throughout the manual, change bars in the margins indicate technical additions and asterisks denote deletions.

Note:
Insert this cover after the manual cover to indicate the updating of the document with Addendum B.

SOFTWARE SUPPORTED

Multics Software Release ~~10.2~~ 11.0

ORDER NUMBER

AW53-04B

March 1984

40126
7.5C484
Printed in U.S.A.

Honeywell

COLLATING INSTRUCTIONS

To update the manual, remove old pages and insert new pages as follows:

Remove	Insert
Title Page, Preface	Title Page, Preface
iii through vi	iii, iv
vii, blank	v, blank
1-3, 1-4	1-3, 1-4
2-1, 2-2	2-1, 2-2
2-7 through 2-22	2-7, 2-8
	2-8.1, blank
	2-9 through 2-22
3-1 through 3-6	3-1 through 3-6
3-9 through 3-12	3-9 through 3-12
3-15 through 3-22	3-15 through 3-22
3-31 through 3-36	3-31 through 3-36
3-41, 3-42	3-41, 3-42
3-55, 3-56	3-55, 3-56
4-9, 4-10	4-9, 4-10
4-21, 4-22	4-21, 4-22
4-25, 4-26	4-25, 4-26
5-5, 5-6	5-5, 5-6
6-7, 6-8	6-7, 6-8
	6-8.1, blank
7-1, 7-2	7-1, 7-2
7-5 through 7-10	7-5 through 7-10
9-3, 9-4	9-3, 9-4
10-1 through 10-14	10-1, blank
11-1 through 11-4	11-1, 11-2
11-5, blank	11-3, blank
12-1 through 12-4	12-1, blank
13-1 through 13-4	13-1 through 13-4
14-15 through 14-18	14-15 through 14-18
A-1 through A-16	A-1 through A-14
F-1 through F-8	F-1 through F-10
F-9, blank	F-11, blank
i-1 through i-6	i-1 through i-4
	i-5, blank
TP Remarks Form	TP Remarks Form

File No.: 1L13, 1U13

3/84

AW53-04B

MULTICS RELATIONAL DATA STORE REFERENCE MANUAL ADDENDUM D

SUBJECT

Changes to the Manual

SPECIAL INSTRUCTIONS

This is the fourth addendum to the AW53-04 dated September 1981. Refer to the Preface for "Significant Changes." Insert the attached pages into the manual according to the collating instructions on the back of this cover. Change bars in the margins indicate technical additions and asterisks denote deletions.

Note:

Insert this cover after the manual cover to indicate the updating of the document with Addendum D.

SOFTWARE SUPPORTED

Multics Software Release 12.0

ORDER NUMBER

AW53-04D

December 1986

47028
187
Printed in U.S.A.

Honeywell

COLLATING INSTRUCTIONS

To update the manual, remove old pages and insert new pages as follows:

REMOVE	INSERT
TP, Preface	TP, Preface
iii through vi	iii through vi vii, blank
3-3, 3-4 3-39 through 3-42	3-3, 3-4 3-39 through 3-42
4-9, 4-9.1 4-9.2, 4-10	4-9, 4-9.1 4-9.2, 4-10
9-23 through 9-26	9-23 through 9-26
11-1, blank	11-1, blank
14-1 through 14-22	14-1 through 14-30 14-31, blank
E-1, E-2	E-1, blank E-1.1, E-2
i-1 through i-5	i-1 through i-4 i-5, blank

File No.: 1L13

12/86

AW53-04D

INDEX

HONEYWELL INFORMATION SYSTEMS
Technical Publications Remarks Form

TITLE: MULTICS RELATIONAL DATA STORE REFERENCE MANUAL ADDENDUM D

ORDER NO. AW53-04D

DATED DECEMBER 1986

ERRORS IN PUBLICATION

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Your comments will be investigated by appropriate technical personnel and action will be taken as required. Receipt of all forms will be acknowledged; however, if you require a detailed reply, check here. ☐

PLEASE FILL IN COMPLETE ADDRESS BELOW.

FROM: NAME _____ DATE _____

TITLE _____

COMPANY _____

ADDRESS _____

_____

CUT ALONG LINE

CUT ALONG LINE

PLEASE FOLD AND TAPE–
NOTE: U.S. Postal Service will not deliver stapled forms

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 39531 WALTHAM, MA 02154

POSTAGE WILL BE PAID BY ADDRESSEE

HONEYWELL INFORMATION SYSTEMS
200 SMITH STREET
WALTHAM, MA 02154

ATTN: PUBLICATIONS, MS486

Honeywell

Honeywell

Honeywell Information Systems
In the U.S.A.: 200 Smith Street, MS 486, Waltham, Massachusetts 02154
In Canada: 155 Gordon Baker Road, Willowdale, Ontario M2H 3N7
In the U.K.: Great West Road, Brentford, Middlesex TW8 9DH
In Australia: 124 Walker Street, North Sydney, N.S.W. 2060
In Mexico: Avenida Nuevo Leon 250, Mexico 11, D.F.

32950, 7.5C1081, Printed in U.S.A.

AW53-04