Roger Roach   PDO

MULTICS TECHNICAL BULLETIN   MTB- 400

To: Distribution

From: Leroy M. Brown

Date: December 7, 1978

Subject: The Multics Data Dictionary

INTRODUCTION

The purpose of this document is to describe the Multics Data Dictionary (MDD). It will define the terminology, commands, and subroutine interface associated with the MDD.

A Data Dictionary is a tool for the management and control of the data resource of an enterprise. It can also be thought of as a tool used to list, describe, and locate each data element in an enterprise. It provides a method to store in a central location all definitions of data within an enterprise together with their attributes for the purpose of controlling how data is created and used. In this manner the total collection of data on which an enterprise depends is improved. A data dictionary is a basic tool within the database environment that assists management, database administrators, analysts, and application programmers in effectively planning, controlling, and evaluating the collection of data resources.

DESCRIPTION

The Multics Data Dictionary (MDD) is a free standing data dictionary. That is, all input to the dictionary is manual. The Data Administrator will be responsible for the integrity of the data. However, MDD is designed so that it could eventually evolve into an integrated dictionary.

The Data Dictionary Manager (DDM) provides the user access to the data dictionary via subroutine calls. The subroutines can be called from any programming language which supports a subroutine call. The subroutine interface allows the user to store, retrieve, modify, and delete data in the data dictionary.

The DDM also supports a dictionary query language. The MDD commands allow the user to perform "ad hoc" queries on the data dictionary. It also provides for creation and maintenance of the data dictionary. The MDD provides several reports on entities in the data dictionary.

BENEFITS

The MDD provides the user with a number of capabilities. These capabilities can be very beneficial when used effectively throughout an an enterprise.

The MDD can be used to help reduce unplanned redundancies in an enterprise. If a programmer needs to name an entity and doesn't know that a name exists for that entity, the entity will be renamed. This creates redundant and inconsistant data in the enterprise. The MDD provides a centralized data base of all data names, descriptions, and usage.

In system development MDD provides the means for all components of a project to speak the same language. The components will become more compatible. This will save time in the system development process.

System maintenance time can be kept at a minimum. All the data names will be documented in the dictionary.

The system can be audited to see where various entities are used. The system can keep track of which entities are used by a particular entity and which entities use the entity.

DATA ADMINISTRATOR

The Data Administrator (DA) shall be responsible for maintaining the data dictionary. All input to the data dictionary should be channeled through the DA.

The DA sets the classifications and statuses of the entities. He then sees that the statuses are kept up to date. The DA is responsible for recovery procedures and backup files for the data dictionary.

Throughout this document we refer to the user. The DA can be one or a group of users. The DA is the only user that modifies the data dictionary.

ATTRIBUTES

Name

The name attribute is the dictionary term for an entity. It can have a length of 1 to 32 non-blank characters. The valid characters are alphabetic, numeric, hyphen, and the underscore. The first character must be alphabetic. Though an entity may have many names in an enterprise, it should only be referred to with one dictionary name. The dictionary name need not be the name used in the enterprise. The entity's name must always be given when storing an entity into the data dictionary.

Description

The description attribute is an English language definition of the entity. It can be up to 256 characters in length. Any characters may be used in the description.

The description entity can be searched on keywords. When using the Where Clause, the kwid (keyword in description) function is used to refer to keyword strings in the description of entities in the data dictionary. Each keyword string can be up to 32 characters in length.

State

The state attribute is the curent usage mode of an entity . It is a 15 character string defining the status of the entity. During development an entity could be in a test state. After the development process is complete and the entity is being used, this could be considered production state. The valid states of an entity are defined by the Data Administrator.

Owner

The owner attribute is the person or group that is responsible for the entity. The owner is a 1 to 32 character string which may be a person or group name. When storing a name with embedded blanks, the name must be enclosed in quotes. The owner of a database would be the data base administrator.

Loc

The loc attribute is the location (pathname, cat/file, etc.) of the entity. A loc name can be up to 168 characters.

Validator

The validator attribute is the location of a validation procedure for an item. It can be up to 168 characters in length.

encode

The encode attribute is the location of the procedure used to encode the value of the item entity.

decode

The decode attribute is the location of a procedure used to decode the values of the item entity.

Data type

The data_type attribute is a pl1 description of the internal representation of a data item i.e. character, fixed bin, fixed dec, etc.

File size

The file_size attribute is the number of words currently in a file. The file size is an integer number x such that: 0<=x<=2**71

Access method

The Access_method attribute indicates the method in which a file is accessed i.e. sequential, indexed, random, etc. It is a 1 to 15 character string specifying an access method type. If the entity is a database the access_method is the DBMS.

Language

The language attribute is the type of programming language a module or program is written in. It is a 1 to 15 character string.

Date created

The date_created attribute is the date the entity was created. The input is of the forms acceptable by_ the convert_date_to_binary_ subroutine.

Date modified

The date_modified attribute is the date which the entity was last updated. The input is of the forms acceptable by the convert_date_to_binary subroutine.

Class

The class attribute is a name given to an association of entities. All of the entities associated with payroll belong to the payroll class. An entity may belong to any number of classes. The class is a 1 to 15 character string.

Type

The type attribute is the name of the kind of entity. It can only be one of the valid entity types as discussed in the section on entities.

Usage

The usage attribute is the name where an entity's alias name is used. It is a 1 to 32 character string.

Index flag

The index_flag attribute specifies whether or not an item is an index. When the item entity is the child of a link relation, the user can denote whether or not it is an index. If the item is an index, the string "yes" follows the item's name. If not the string "no" follows the name. The default is "no".

Alias name

The alias_name attribute is the name of the entity as used in the places specified by the usage attribute.

SYSTEM COMPONENTS

ENTITIES

Name: item,i

The item entity is the lowest component of data which may be defined in the data dictionary.

Attributes

(name, desc, state, owner, data_type, val, encode, decode)

Where:

1. name
   is the name by which the item is referred to in the dictionary.

2. desc
   is a description of the item.

3. state
   is the mode of usage which the item is in.

4. owner
   is the person responsible for maintenance.

5. data_type
   is a pl1 description of the internal representation of the data item.

6. val
   is the location of the validation procedure.

7. encode
   is the location of the encoding procedure for the values of the item.

8. decode
   is the location of the decoding procedure for the item.

Name: group,g

A group is composed of a set of related items and/or groups.

<u>Attributes</u>

(name, desc, state, owner)

where:

1. name
   is the name by which the group is referred to in the dictionary.

2. desc
   is a description of the group.

3. state
   is the mode of usage which the group is in.

4. owner
   is the person responsible for maintenance.

<u>Name</u>: record,rec

A record is a collection of groups and/or items that are in some way related.

<u>Attributes</u>

(name, desc, state, owner)

where:

1. name
   is the name by which the record is referred to in the dictionary.

2. desc
   is a description of the record.

3. state

is the mode of usage which the record is in.

4. owner
is the person responsible for maintenance.

Name: file,f

A file is a collection of one or more records.

Attributes

(name, desc, state, owner, date_created, date_modifies, loc, file_size, access_method, file_type)

where:

1. name
is the name by which the file is referred to in the dictionary.

2. desc
is a description of the file.

3. state
is the mode of usage which the file is in.

4. owner
is the person responsible for maintenance .

5. date_created
is the date the file was created.

6. date_modified
is the date the file was last updated.

7. loc
is the loc name of the file.

8. file_size
is the number of words currently in the file.

9. file_type
is the type of file (sequential, random, etc.).

10. access_method

is the method in which the file is accessed. (vfile, ansi_tape, etc.)

Name: database_view,dbv

The database_view is the object file produced by a source file that is a subset of a database. A subset of the database would be a subschema, data_submodel, etc.

Attributes

(name, desc, state, owner, date_created, date_modified, loc)

where:

1. name
   is the name by which the database_view is referred to in the dictionary.

2. desc
   is a description of the database_view.

3. state
   is the mode of usage which the database_view is in.

4. owner
   is the person responsible for maintenance .

5. date_created
   is the date the database_view was created.

6. date_modified
   is the date of the last update to the database_view.

7. loc
   is the location of the object file for the database_view.

Notes

Files which are linked to the database_view include the source file that describes a subset of the database and other files associated with that subset.

Name: database,db

The database entity is a collection of records which are physically placed and retrieved by a data base management system.

Attributes

(name, desc, state, owner, date_created, date_modified, loc)

where:

1. name
   is the name by which the database is referred to in the dictionary.

2. desc
   is a description of the database.

3. state
   is the mode of usage which the database is in.

4. owner
   is the person responsible for maintenance .

5. date_created
   is the date the database was created.

6. date_modified
   is the date the database was last updated.

7. loc
   is the location of the database.

Notes

Files that are linked to the database entity include the database source file
and other files associated with the database.

Name: report,rpt

The report entity is an an output which is to be generated by a program.

Attributes

(name, desc, state, owner)

where:

1. name
   is the name by which the report is referred to in the dictionary.

2. desc
   is a description of the report.

3. state
   is the mode of usage which the report is in.

4. owner
   is the person responsible for maintenance .

Name: module,m

The module is a group of computer instructions that perform a function that is called by a program.

Attributes

(name, desc, state, owner, date_created, date_modified, loc)

where:

1. name

is the name by which the module is referred to in the dictionary.

2. desc
   is a description of the module.

3. state
   is the mode of usage which the module is in.

4. owner
   is the person responsible for maintenance .

5. date_created
   is the date the module was created.

6. date_modified
   is the date of the last update to the module.

7. loc
   is the loc name of the module.

Name: program,p

The program entity is a collection of processable code that manipulates data.

Attributes:

(name, desc, state, owner, date_created, date_modified, loc, language)

where:

1. name
   is the name by which the program is referred to in the data dictionary.

2. desc
   is a description of the program.

3. status
   is the mode of usage which the program is in.

4. owner
   is the person responsible for maintenance.

5. date_created
   is the date the program was written.

6. date_modified
   is the date the program was last updated.

7. loc
   is the loc name of the program.

8. language
   is the source language which the program is written in.

Name: system,s

The system is a collection of programs and modules that accomplish a major function.

Attributes

(name, desc, state, owner)

where:

1. name
   is the name by which the system is referred to in the dictionary.

2. desc
   is a description of the system.

3. state
   is the mode of usage which the system is in.

4. owner
   is the person responsible for maintenance .

Name: user,u

The user entity is a person or group which interacts with an entity.

Attributes

(name, desc)

where:

1. name
   is the name by which the user is referred to in the dictionary.

2. desc
   is a description of the entity.

Notes

The user entity may have any entity linked to it. It is the only entity which may have all entity types linked to it.

RELATIONS

Name: alias, a

Alias names are various names which rename an entity within an enterprise.

Attributes

(name, type,usage,alias_name)

where:

1. name
   is the dictionary name of the entity.

2. type
   is the type of entity.

3. usage
   is where the alias name is used.

4. alias_name
   is the name that is used.

Name: class

The class relation assigns an entity to one or more categories.

Attributes

(name, type, class)

where:

1. name
   is the dictionary name of the entity.

2. type
   is the type of entity.

3. class
   is the name of a department or category to which an entity belongs.

Name: link

The link relation allows the user to link entities to other entities. It links a child entity to an parent entity.

Attributes

(name1, type1, name2, type2, index_flag)

where:

1. name1
   is the entity name for the parent entity.

2. name2
   is the entity name for the child entity.

3. type1
   is the entity type for the parent entity.

4. type2
   is the entity type for the child.

5. index_flag
   is a code which denotes an index. It is used when item is the member entity, otherwise it is ignored.

** See table-1 on following page.

Table-1: Entity Linkage Table

	item	group	record	file	database view	database	module	report	program	system	user
item											
group	x	x									
record	x	x									
file		x						x			
database view				x					x		
database			x	x				x			
module											
report											
program							x	x			
system							x		x		
user	x	x	x	x	x	x	x	x	x	x	

SYSTEM FUNCTIONS

WHERE CLAUSE

The modify, delete, and print functions may specify a where clause. The where clause sets the conditions required to identify an entity. The where clause consist of terms of the form attribute name followed by a relational operator followed by a string or constant. This can be expanded by using parentheses and the logical operators.

valid relational operators are:

=	equal to
^=	not equal to
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to

valid logical operators are:

&	logical and
¦	logical or
^	logical not

The descriptor attribute is denoted by the name kwid(keyword in descriptor). The user can specify an entity with certain keywords in the descriptor.

** For a list of entities and associated attributes see Table-2.

Table-2: Entities and Attributes

	name	desc	state	owner	date_created	date_modified	loc	data_type	val	encode	decode	file_size	file_type	access_method	language
item	x	x	x	x				x	x	x	x				
group	x	x	x	x											
record	x	x	x	x											
file	x	x	x	x	x	x	x					x	x	x	
database view	x	x	x	x	x	x	x								
database	x	x	x	x	x	x	x								
module	x	x	x	x	x	x	x								x
report	x	x	x	x											
program	x	x	x	x	x	x	x								x
system	x	x	x	x	x	x									
user	x	x													

COMMAND DESCRIPTION

Name: multics_dd, mdd

The multics_dd command is used to enter the data dictionary facility. The mdd commands allows retrieval and update operations to be performed on a data dictionary.

Usage

```
multics_dd
```

where path is the pathname of an input file with data dictionary requests.

After the data dictionary facility has been entered, the user can create a data dictionary or open a previously opened data dictionary. Only one data dictionary may be open to a process.

Multics dd Requests

The remainder of this section contains descriptions and examples of the multics_dd requests.

Request: store, s

This command allows the user to store entities into the Multics Data Dictionary. The entities may be stored one at a time or a file of entities may be stored.

Usage

```
    store entity_type {string1...stringn} {control_args}
```

where:

1. entity_type

is one of the entities as described in the section on entities.

2. stringi

   are the values of the attributes to be stored.

3. control_args

   -file path, -f path

   A file of the same entities that are to be stored. The attributes names are not needed, but the attributes must be in the order of the attributes as shown for the entity in the section on entities.

   -delimiter char, -d char

   Separates each attribute of the entity found in the file by the character char.

Notes

The name attribute must always be given. After the user has given the attributes he wishes to store, the system will prompt him for the alias, class, and link relations.

If no attributes are given, the system will prompt him for each attribute and relation. Each attribute that is not given a value is assigned a blank value before being stored.

Examples:
(system output is underlined)

1. Store the item entity grade into the dictionary.

```
store i
Name?  grade
state?  test
desc?  "The grade is part of the student's record.  It shows his level
of achievement."
owner?
val?
data type? char (1)
alias?
class?  student_records
```

2. Store the record entity employee_rec into the dictionary.

```
store rcd employee_rec p "" "Jones"
alias? emprec in fortran employee in programa
```

```
class? accounting payroll benefits
link? -items employee_name"yes" badge_id cost_center -group address
```

Request: print, p

This command allows the user to print some or all of the attributes of an entity at the terminal.

Usage

```
print type {attr1...attrn} {where_clause} {control_args}
```

where:

1. type
   is the type of entity or relation.

2. attri
   is the name of the attribute the user desires to print.

3. where_clause
   is an expression which specifies the conditions required to identify the entities the user wishes to print.

4. control_arg
   -output_file path, -of path where path is the pathname of a file to which the output is written.

Example:

1. Print the records in production status that have the keyword ball in the description.

   ```
   print rcd  -where (state=p) & (kwid=ball)
   ```

2. print the name of all records in the dictionary.

   ```
   print rcd name
   ```

Request: delete, d

This command allows the user to delete an entity or relation from the dictionary.

<u>Usage</u>

```
   delete  type where_clause
```

where

1. type
   is the type of entity or relation.

2. where_clause
   is an expression which specifies the conditions required to identify the entities or relations the user wishes to delete.

<u>Notes</u>

If an entity is deleted, all relations associated with that entity are also deleted.

Examples:

Delete all of Pearson's programs.

```
    delete p -where owner=Pearson
```

<u>Request</u>: modify,m

This command allows the user to modify attributes of an entity or add relations to an entity.

<u>Usage</u>

```
   modify type {attr1...attrn} {rel1...} where_clause
```

where:

1. type
   is the type of entity to be modified or relation added.

2. attri
   is the name of the attributes to be modified.

3. reli

is the name of the relation to be added.

4. where_clause

is an expression which specifies the conditions required to identify the entities the user wishes to modify.

Notes

The system will prompt the user for values of the attributes to be modified. A list of all the entity names will be printed to ask the user if the modifications are ok before changes are made.

Examples:
(system output is underlined)

Change the owner and the status of the data model entity db1.

```
modify db owner state -where name=db1;
new owner?  Dorsett
new state?  p
```

Request: create_dd,cr

This command allows the user to create a data dictionary.

Usage

```
cr path
```

Where path is the pathname of the data dictionary.

Request: open, o

This command allows the user to open a data dictionary.

Usage

```
open path
```

where pathname is the pathname of the data dictionary.

Notes

Only one data dictionary may be open to a process.

Request: close, c

This command allows the user to close a previously open data dictionary.

Usage

```
close
```

Request, q: quit, q

This command allows the user to exit the MDD facility.

Usage:

```
quit
```

Request: execute, e

This command allows the user to execute multics commands from within the MDD facility.

Usage

```
execute command
```

Where command is one of the multics commands.

Reports

Request: locate,l

This command allows the user to locate the entity types of a given name.

Usage

```
   locate name
```

Where name is a data dictionary name.

Example:

Locate the name temp1

```
locate temp1
```

Entity type

```
record
file
database_view
```

Request: write_glossary, wg

The write_glossary command will write a report of the entities. It will give the name, description, status, and classification of each entity.

Usage

```
   write_glossary {entity_type1 ... entity_typen} control_arg
```

where:

```
1.  entity_typei
            is the type of entity.

2.  control_arg
            -output_file path, -of path
                 where path is the pathname for the output.
```

Notes

If no type is specified, all types are printed.

Format of a glossary output can be found in appendix A

Request: print_catalog, pc

The list_catalog command gives a report on an entity or group of entities.

Usage

```
   print_catalog entity_type {-where_clause} {name1...namen} {control_args
```

where:

1. entity_type
   is the type of entity.

2. where_clause
   is an expression which specifies the conditions required to identify the entities the user whiches to print the catalog of.

3. namei
   is the dictionary name of the entity.

4. control_args
   -output_file path -of path
   where path is the name of a file to which the output is directed.

Information given:

1. Which entities are members
2. Type of member
3. Which entities are owners
4. type of owners
5. alias names used

6. where these names are used

Example of a catalog output for employee_rec record:

```
References                          type
name                                item
address                             group
pay_no                              item

Referenced-by                       type
med_insurance                       file
employee_profile                    file

Alias                               Where
emprec                              fortran
employee                            database_1
```

SUBROUTINES

Entry: mdd_$store

This entry allows the user to store data into the data dictionary.

Usage

```
     declare mdd_$store entry options (variable);

     call mdd_$store (type, attr1,...,attrn, code);
```

where:

1. type (input) (fixed bin (35))
   is a code which specifies the type of entity or relation to be stored.

2. attri (input) (char(*))
   is an attribute peculiar to the type of entity as decribed in the section on entity type. If relation type, this is an attribute associated with the relation type as described in the section on relations.

3. code (input) (fixed bin (35))
   is a standard status code.

Entry: mdd_$retrieve

This entry allows the user to retrieve data from the dictionary.

Usage

```
     declare mdd_$retrieve entry options (variable);

     call mdd_$retrieve (type, attr1,...,attrn, where_clause,
          code);
```

where:

1. type (input) (fixed bin (35))
   is a code which specifies the type of entity or relation to be retrieved.

2. attri (output) (char(*))
   is the attributes peculiar to the type of entity as described in the section on entity type. If relation type, this is the attribute associated with the relation type as described on the section on relations.

3. where_clause (input) (char (*))
   is an expression which specifies the conditions required to identify the entities which the user desires to access.

4. code (input) (fixed bin (35))
   is a standard status code.

<u>Entry</u>: mdd_$locate

This entry allows the user to find the dictionary type of a a given entity name.

<u>Usage</u>

```
declare mdd_$locate (char (32), char (16), fixed bin (35));

call mdd_$locate (name, type, code);
```

where:

1. name (input)
   is the dictionary name the user whiches to locate.

2. type (output)
   is the type of entity.

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

Entry: mdd_$modify

This entry allows the user to modify data in the dictionary.

Usage

```
declare mdd_$modify entry options (variable);

call mdd_$modify  (type,  string1,...,stringn, where_clause,
     code);
```

where:

1. type (input) (fixed bin (35))
   is a code which specifies the type of entity or relation to be modified.

2. attri (input) (char (*))
   is the attribute peculiar to the type of entity as decribed in the section on entity type. If relation type, this is the attribute associated with the relation type as described on the section on relations.

3. where_clause (input) (char (*))
   is an expression which specifies the conditions required to identify the entities which the user desires to access.

4. code (output) (fixed bin (35))
   is a standard status code.

Entry: mdd_$delete

This entry allows the user to delete data from the dictionary.

Usage

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

```
     call mdd_$delete (type, where_clause, code);
```

where:

1. type (input) (fixed bin (35))
   is a code which specifies the type of entity or relation to be deleted.

2. where_clause (input) (char (*))
   is an expression which specifies the conditions required to identify the entities which the user desires to access.

3. code (output) (fixed bin (35))
   is a standard status code.

<u>Entry</u>: mdd_$open

This entry allows the user to open a data dictionary. Only one data dictionary may be open for each process.

<u>Usage</u>

```
     declare mdd_$open entry (char(168),fixed bin (35), fixed bin
          (35));

     call mdd_$open (path, code)
```

where:

1. path (input) (char (168))
   is the dictionary's pathname.

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

<u>Entry</u>: mdd_$close

This entry closes a previously opened data dictionary.

<u>Usage</u>

```
    declare  mdd_$close entry(fixed bin (35));

    call mdd_$close(code);
```

Where code is a standard status code.

Appendix A

DATA DICTIONARY GLOSSARY

ITEMS

name	class	state	description
pay-no	accounting payroll	p	The employee's pay number
phone_no	personnel	t	The home phone numbers of the employees
.	.	.	.
.	.	.	.
.	.	.	.

GROUPS

name	class	state	description
address	personnel	p	This is the employee's address. It is the mailing address for the employee's checks and other confidential information.
cust_address	accounting	p	This is the customer's address. It is the billin address for the customer.
.	.	.	.
.	.	.	.
.	.	.	.