NBSIR 85-3165

Using the Information Resource Dictionary System Command Language

Alan Goldfine

U.S. DEPARTMENT OF COMMERCE
National Bureau of Standards
Center for Programming Science and Technology
Institute for Computer Sciences and Technology
Gaithersburg, MD 20899

April 1985

U.S. DEPARTMENT OF COMMERCE

NATIONAL BUREAU OF STANDARDS

NBSIR 85-3165

USING THE INFORMATION RESOURCE DICTIONARY SYSTEM COMMAND LANGUAGE

Alan Goldfine

U.S. DEPARTMENT OF COMMERCE
National Bureau of Standards
Center for Programming Science and Technology
Institute for Computer Sciences and Technology
Gaithersburg, MD 20899

April 1985

USING THE INFORMATION RESOURCE DICTIONARY SYSTEM COMMAND LANGUAGE

Alan Goldfine

This document introduces and provides examples of the Command Language of the draft proposed Information Resource Dictionary System (IRDS). A dictionary maintained by the U.S. Air Force is defined in the IRDS and used as a continuing example throughout the document. The dictionary is populated, manipulated, and reported on using the precise syntax of the Command Language. An appendix to the document provides a complete listing of the creation of the example. Other appendices provide indices of all command appearances and all clause appearances.

Key words: command language; data dictionary; data dictionary system; data dictionary system standard; example book; Information Resource Dictionary System; IRDS.

ACKNOWLEDGEMENTS

We wish to thank the U.S. Air Force Air Staff Programs and Financial Systems Group for their permission to use the Air Staff Codes and Descriptions (ASCAD) data dictionary as the continuing example in this book. We also wish to acknowledge the assistance of Frank Spielman of ICST/NBS, formerly of the Air Force, for bringing the ASCAD dictionary to our attention and explaining its characteristics.

TABLE OF CONTENTS

1. INTRODUCTION

This document is designed to accompany the specification of the Information Resource Dictionary System (IRDS) [1], [2], [3], [4].

The Core IRDS specifies two direct user interfaces: a menu-driven "Panel" Interface, designed to support interactive processing, and a Command Language that may be used in either a batch or interactive mode. This volume introduces and provides examples for the Command Language.

Although the Command Language is completely described in the Core IRDS Specifications, the Backus-Naur notation used is not designed for tutorial purposes. In this document, we illustrate a "real world" Information Resource Dictionary example, and show how such a dictionary could be populated, manipulated, and reported on using the Command Language.

We assume that readers of this document will be referring to the Command Language syntax in the IRDS Specifications, and that they are familiar with the contents of _A Technical Overview of the Information Resource Dictionary System_ [5].

2. THE GLOBAL EXAMPLE

2.1 DESCRIPTION

We will base our global example on the dictionary maintained by the U.S. Air Force to support its Air Staff Codes and Descriptions (ASCAD) application. The database for the ASCAD application contains "all common (corporate) data elements which are codes and their respective descriptions." Figure 2-1 illustrates the overall structure of the ASCAD dictionary, expressed in terms of the Core IRDS System-Standard Schema.

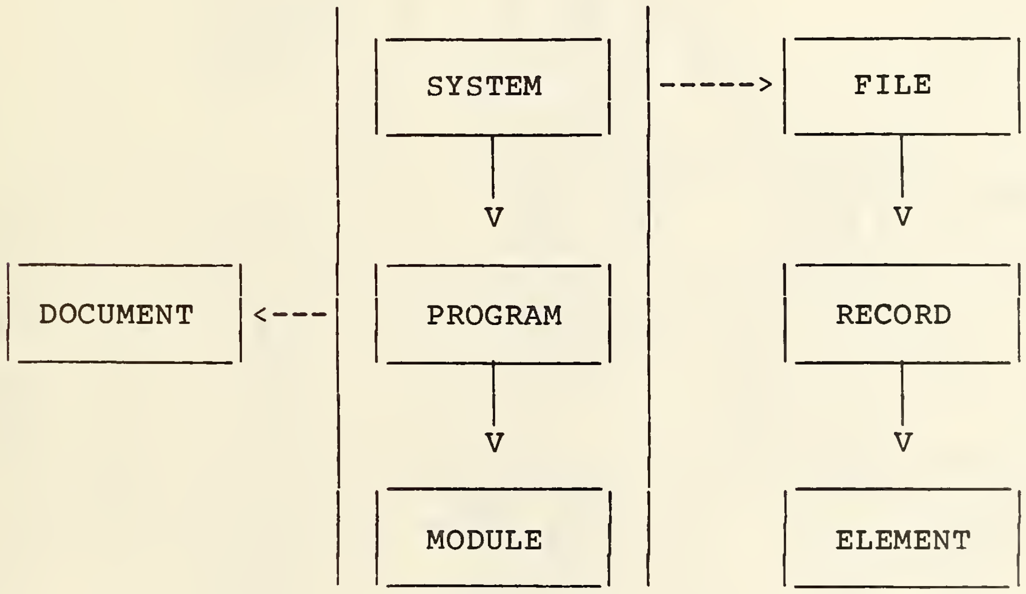

Figure 2-1. Overall Structure of IRDS Application

The full ASCAD dictionary contains many thousands of entities, but our examples will be restricted to a small subset of this. We will populate the (initially empty) dictionary with the collection of 34 entities illustrated by Figure 2-2.

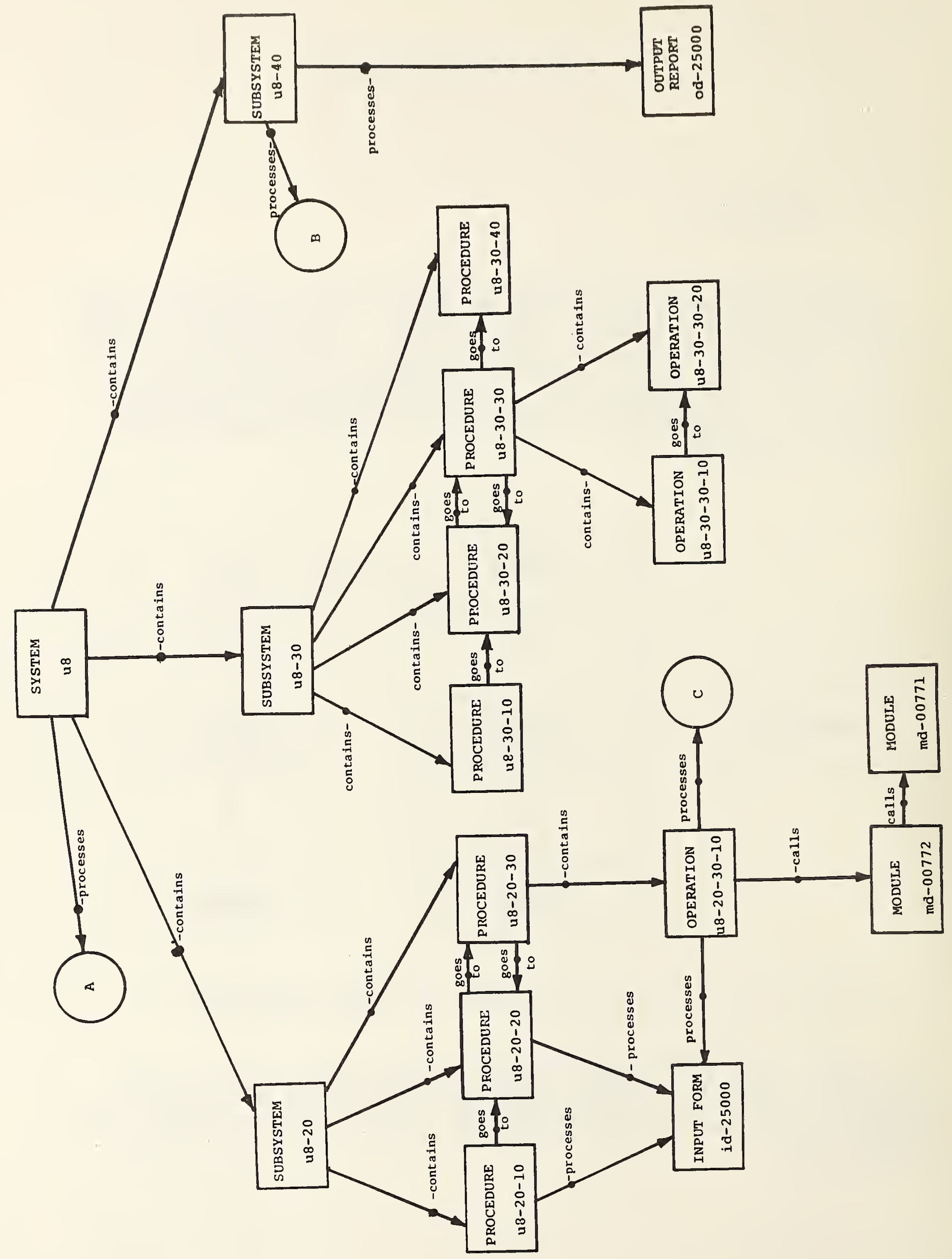

Figure 2-2 (Part 1)

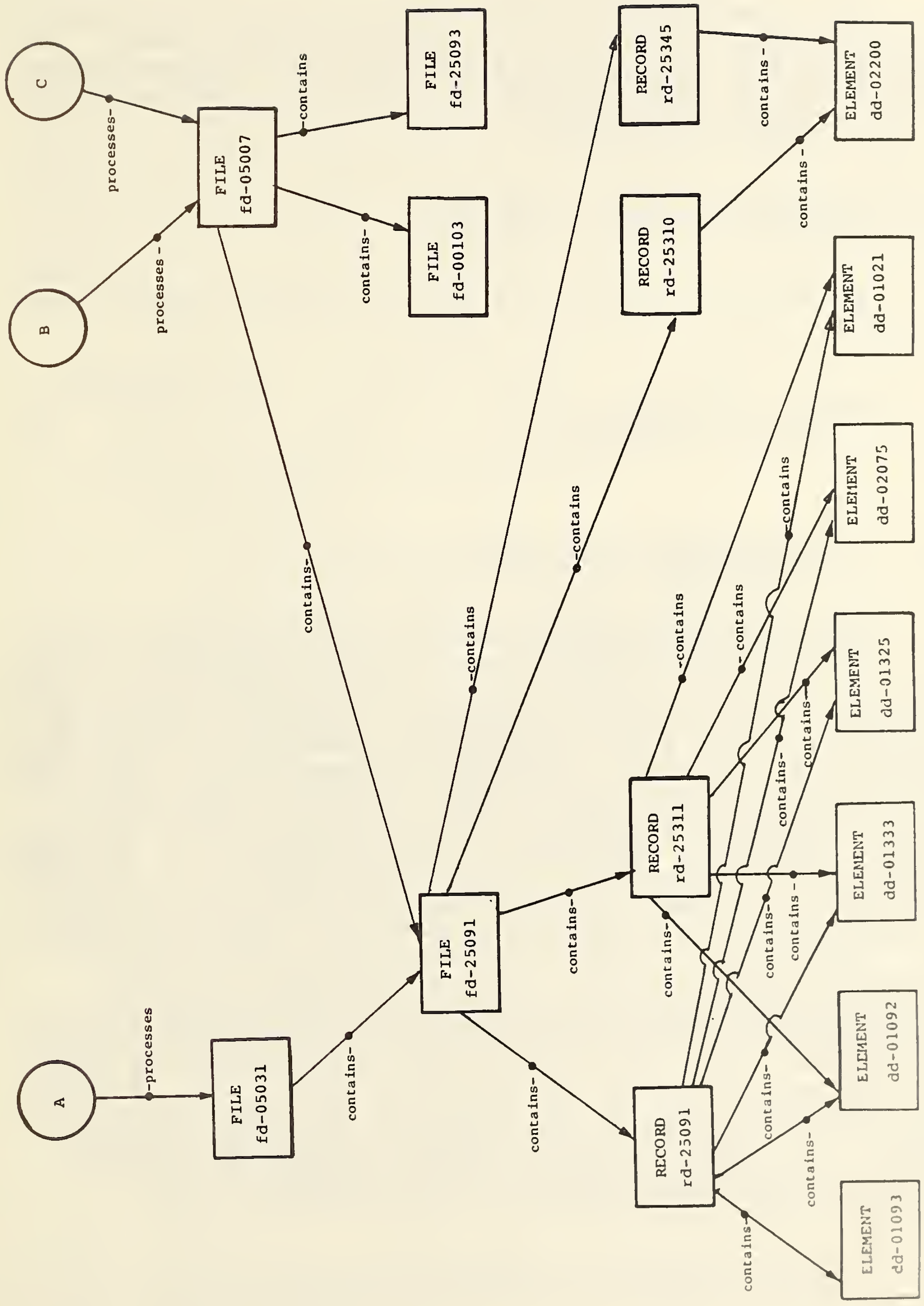

Figure 2-2 (Part 2)

Note that many of the entities in this diagram appear to be of types not in the Core System-Standard Schema. We will initially assign these entities to Core System-Standard Schema entity-types. In Chapter 4, we will demonstrate how to define new entity-types.

2.2 CREATING AN EMPTY DICTIONARY

We begin by creating the empty dictionary Example, and associating with it the Core System-Standard Schema:

```
CREATE DICTIONARY  Example
  SCHEMA IS STANDARD;
```

2.3 POPULATING THE DICTIONARY

Although our principal concern in this document is to provide examples of IRDS Command Language usage, we will do so in a way that demonstrates the basic capabilities of such a system, and that also illustrates reasonable dictionary construction and usage techniques. Therefore, we will populate the dictionary in a largely top-down manner, first delineating the "broad picture" of the overall application structure, then returning to fill in the detailed properties of the individual components. First, we sketch in the ASCAD system/subsystem/procedure hierarchy. The flow of execution and control between components of this "process" hierarchy (and associated programs and modules) can then be documented. We next outline the structure of the application's data, by documenting its file/record/element hierarchy. These skeletal data descriptions are then integrated into the system hierarchy by specifying the appropriate usage information. Similarly, we introduce the descriptions of input and output documents used by the application. Finally, we fill in the gaps by describing the user-specified attributes of the application's individual entities. (Several audit attributes are automatically assigned by the IRDS directly upon establishment of an entity.)

2.3.1 The Overall Application System Structure.

As we see from Figure 2-2, our subset of the ASCAD database contains one system, three subsystems, and seven procedures. Using the Core IRDS System-Standard Schema, we represent each of these with the entity-type SYSTEM. In Chapter 4, we will show how the dictionary can be customized to explicitly represent the unique characteristics of subsystems and procedures. We begin by creating u8, the entity representing the entire application system. We will define description and external security requirement attributes for u8:

```
ADD ENTITY u8
  ENTITY-TYPE = SYSTEM
  DESCRIPTIVE-NAME =
    ASCAD-Database-Information-System
  WITH ATTRIBUTES
    DESCRIPTION =
        "This system provides the necessary tools
      for maintaining the Air Staff Codes and
      Descriptions (ASCAD) Database. The ASCAD
      Database contains all common (corporate) data
      elements which are codes and their respective
      descriptions. The tools provide the
      capability:
        1. To control access to the database
          a. single record at a time
          b. groups of records
        2. To update the tables in the database
        3. To produce reports from the database
        4. To create tapes containing database
           information
        5. To display information online.",
    SECURITY = "datamgr";
```

u8 contains three subsystems; we will define u8-20 here, and leave the definitions of u8-30 and u8-40 to the complete command listing in Appendix A.

```
ADD ENTITY  u8-20  ENTITY-TYPE = SYSTEM
  DESCRIPTIVE-NAME = ASCAD-Database-Update
  WITH ATTRIBUTES
    DESCRIPTION  (START = 100  INCREMENT = 10)
    =
     "This subsystem provides the capability for
    the Air Staff to update the contents of the
    ASCAD Database.",
     SYSTEM-CATEGORY = "subsystem",
     SECURITY = "datamgr";
```

The attribute-type SYSTEM-CATEGORY allows these SYSTEM entities to be identified as "subsystems". In describing text attributes, the starting line number and increment value can be specified to override the IRDS default.

Since the IRDS views a relationship, not as an attribute of a single entity, but as a totally different structure that links two distinct entities, we must use the ADD RELATIONSHIP command to connect u8 with its three subsystems:

```
ADD RELATIONSHIP
  u8  SYSTEM-CONTAINS-SYSTEM  u8-20;
```

Likewise with u8 containing u8-30 and u8-40.

In a similar manner, we define u8-20-10, u8-20-20, and u8-20-30 as PROCEDUREs contained in u8-20, and u8-30-10, u8-30-20, u8-30-30, and u8-30-40 as PROCEDUREs contained in u8-30 (see Appendix A).

PROCEDURE u8-20-30 contains PROGRAM u8-20-30-10, which calls MODULE md-00772, which in turn calls MODULE md-00771:

```
ADD ENTITY  u8-20-30-10
  ENTITY-TYPE = PROGRAM
    DESCRIPTIVE-NAME = ASCAD-Update;
------------------------------------------------------
ADD ENTITY  md-00772
  ENTITY-TYPE = MODULE
    DESCRIPTIVE-NAME = generalized-ASCAD-update;
------------------------------------------------------
ADD ENTITY md-00771 ENTITY-TYPE = MODULE
  DESCRIPTIVE-NAME = generalized-mrds;
------------------------------------------------------
ADD RELATIONSHIP
  u8-20-30-10 PROGRAM-CALLS-MODULE md-00772;
------------------------------------------------------
ADD RELATIONSHIP
  md-00772 MODULE-CALLS-MODULE md-00771;
------------------------------------------------------
ADD RELATIONSHIP
  u8-20-30 SYSTEM-CONTAINS-PROGRAM u8-20-30-10;
```

The IRDS uses the GOES-TO relationship-type to document instances where there is a known flow of execution between two PROCESS entities. Thus:

```
ADD RELATIONSHIP
  u8-20-10 SYSTEM-GOES-TO-SYSTEM u8-20-20;
```
```
ADD RELATIONSHIP
  u8-20-20 SYSTEM-GOES-TO-SYSTEM u8-20-30;
```
```
ADD RELATIONSHIP
  u8-20-30 SYSTEM-GOES-TO-SYSTEM u8-20-20;
```
```
ADD RELATIONSHIP
  u8-30-20 SYSTEM-COMES-FROM-SYSTEM u8-30-10;
```
```
ADD RELATIONSHIP
  u8-30-30 SYSTEM-COMES-FROM-SYSTEM u8-30-20;
```
```
ADD RELATIONSHIP
  u8-30-20 SYSTEM-COMES-FROM-SYSTEM u8-30-30;
```

In the last three commands, we used the SYSTEM-COMES-FROM -SYSTEM inverse name of the SYSTEM-GOES-TO-SYSTEM relationship-type. Note that to use this optional formulation, which is available for all relationship-types (and is really just a convenience for the user), the member entities must be specified in the appropriate order. For example, "x GOES-TO y" is equivalent to "y COMES-FROM x."

An option within commands specifying relationships is the use of the relationship-class-type clause. This alternate formulation allows the user to identify a relationship-type by writing COMES-FROM instead of SYSTEM-COMES-FROM-SYSTEM say, or CONTAINS instead of PROGRAM-CONTAINS-MODULE. This is certainly more convenient, and presents no problem if both member entities have already been defined, since their types will be known to the IRDS. In this case, the user does not have to repeat the information. Thus we have the command:

```
ADD RELATIONSHIP
  u8-30-40 COMES-FROM u8-30-30;
```

On the other hand, if an entity specified as part of a relationship has not been previously defined, its type must be included within the ADD RELATIONSHIP command, in order for the IRDS to have enough information to automatically create the entity. As an example of this syntax, we implicitly define the two PROGRAMs contained within PROCEDURE u8-30-30:

```
-------------------------------------------------------
| ADD RELATIONSHIP                                    |
|   u8-30-30 CONTAINS NEW PROGRAM u8-30-30-10;        |
|-----------------------------------------------------|
| ADD RELATIONSHIP                                    |
|   u8-30-30 CONTAINS NEW PROGRAM u8-30-30-20;        |
-------------------------------------------------------
```

Entities u8-30-30-10 and u8-30-30-20 are now established.

We can then quickly specify:

```
-------------------------------------------------------
| ADD RELATIONSHIP                                    |
|   u8-30-30-10 GOES-TO u8-30-30-20;                  |
-------------------------------------------------------
```

2.3.2 The Data Entities.

The ASCAD application data can be viewed as a FILE/RECORD/ELEMENT hierarchy, containing several levels of FILEs to represent its database structure. FILE fd-05031 contains fd-25091, and FILE fd-05007 contains fd-00103 and fd-25093, as well as fd-25091 (see Figure 2.2). We can declare this using the same techniques as in the previous section (see Appendix A).

Now, FILE fd-25091 contains four RECORDs. We create one, rd-25091:

```
-------------------------------------------------------
| ADD ENTITY rd-25091  ENTITY-TYPE = RECORD           |
|   DESCRIPTIVE-NAME = Countries/States;              |
|-----------------------------------------------------|
| ADD RELATIONSHIP                                    |
|   fd-25091 FILE-CONTAINS-RECORD rd-25091;           |
-------------------------------------------------------
```

We now use rd-25091 as a template for the construction of the other three RECORD entities by employing the COPY ENTITY command:

```
 ---------------------------------------------------------
| COPY ENTITY                                             |
|   rd-25091 WITH RELATIONSHIPS TO  rd-25311              |
|   DESCRIPTIVE-NAME = Countries/States-NK;               |
|---------------------------------------------------------|
| COPY ENTITY                                             |
|   rd-25091 WITH RELS TO rd-25310                        |
|   DNAME = Countries/States-Key;                         |
|---------------------------------------------------------|
| COPY ENTITY                                             |
|   rd-25091 WITH RELS TO rd-25345                        |
|   DNAME = Countries/States-Key-PR;                      |
 ---------------------------------------------------------
```

Since each of the last three RECORDs is contained in the same FILE (fd-25091) as is rd-25091, we were able to use the WITH RELATIONSHIPS option on the COPY ENTITY command.

In our subset of the ASCAD dictionary, RECORD rd-25091 is comprised of the six ELEMENTs dd-01093, dd-01092, dd-01333, dd-01325, dd-02075, and dd-01021. The last five of these ELEMENTs also constitute RECORD rd-25311. Appendix A contains the commands defining these ELEMENTs and their relationships.

We define ELEMENT dd-02200, which is contained in both rd-25310 and rd-25345:

```
 ---------------------------------------------------------
| ADD ENTITY dd-02200                                     |
|   ETYPE = ELE  DNAME = Action-Code;                     |
|---------------------------------------------------------|
| ADD REL rd-25310 CONTAINS dd-02200;                     |
|---------------------------------------------------------|
| ADD REL rd-25345 CONTAINS dd-02200;                     |
 ---------------------------------------------------------
```

In the last six commands, we have used several of the valid abbreviations listed in Section 4.3 of the Core Standard IRDS Specifications [1].

2.3.3 Describing Input and Output Documents.

There are two more entities in our application, DOCUMENTs representing input forms and output reports:

```
 ---------------------------------------------------------
| ADD ENTITY id-25000  ENTITY-TYPE = DOCUMENT             |
|   DESCRIPTIVE-NAME = ASCAD-Table-Change-Request;        |
|---------------------------------------------------------|
ADD ENTITY od-25000  ETYPE = DOC
  DNAME = ASCAD-Table;
```

2.3.4 Completing the Population of the Example.

The process and data hierarchies are linked together, along with the DOCUMENTs, by PROCESS relationships.

```
ADD RELATIONSHIP
  u8 SYSTEM-PROCESSES-FILE fd-05031;
-----------------------------------------------------
ADD REL u8-40 SYSTEM-PROCESSES-FILE fd-05007;
-----------------------------------------------------
ADD REL u8-20-30-10 PROCESSES fd-05007;
-----------------------------------------------------
ADD REL u8-20-10 PROCESSES id-25000;
-----------------------------------------------------
ADD REL u8-20-20 PROCESSES id-25000;
-----------------------------------------------------
ADD REL u8-20-30-10 PROCESSES id-25000;
-----------------------------------------------------
ADD REL od-25000 PROCESSED-BY u8-40;
```

2.3.5 Filling in the Attributes.

Having sketched in the overall structure and interrelationships of the application dictionary, our next step is to go back and use MODIFY ENTITY and MODIFY RELATIONSHIP commands to fill in the attributes of the individual components.

For example, each ELEMENT in the dictionary has associated with it a number of characteristics. We can document, in addition to the ELEMENT´s general description, its data class and external security requirements, among other properties, e.g.:

```
MODIFY ENTITY dd-01093
  WITH ATTRIBUTES
    DESCRIPTION = "A shared data field occupied by
      either cntry-code or state-code",
    SECURITY = "datamgr",
    DATA-CLASS = "alphanumeric",
    IDENTIFICATION-NAMES =
      (ALTERNATE-NAME = "cntry-st-code",
|        ALTERNATE-NAME-CONTEXT = "pl1");                  |
 ----------------------------------------------------------
```

In this command, IDENTIFICATION-NAMES is an example of an attribute-group-type, a sequence of related attribute-types whose attributes are frequently or always used together to document a property of an entity. We are assigning an alternate name, (sometimes referred to as an "alias" or "synonym") to the ELEMENT dd-01093. "Cntry-st-code" is used as the alternate name of dd-01093 in PL-1 programs; if there were FORTRAN programs that referred to dd-01093, the ELEMENT might have an alternate name of CSC093 in that context. Although they are linked together in IDENTIFICATION-NAMES, ALTERNATE-NAME and ALTERNATE-NAME-CONTEXT are individual attribute-types, and attributes of these two types can be defined and accessed separately.

An entity can have several alternate names and, as can be seen from the MOD ENTITY commands for fd-25091, rd-25091, rd-25311, and rd-25345 in Appendix A, alternate names need not be unique in the dictionary, or even for entities of a given type.

A feature of the IRDS is that a relationship between two entities can itself have attributes. For example, it´s useful to be able to specify the relative position of an ELEMENT within a RECORD, e.g.:

```
 ----------------------------------------------------------
| MODIFY RELATIONSHIP rd-25091 CONTAINS dd-01092           |
|   WITH ATTRIBUTES  RELATIVE-POSITION = 3;                |
 ----------------------------------------------------------
```

Appendix A includes all the MODIFY commands necessary to fully specify the attributes of all entities and relationships.

2.3.6 Freezing the Global Example.

Our global application dictionary example is now in place, with the complete specification given by the commands in Appendix A. The example will be considered "frozen," in the sense that any change to its content made by a future example of command usage will be considered local to that example.

2.4 MANIPULATING THE DICTIONARY

2.4.1 Deleting Entities and Relationships.

An entity cannot be deleted from a dictionary if it is a member of a relationship. To delete the DOCUMENT id-25000 say, we must first remove it from all its relationships. The most obvious way is to simply:

```
 ----------------------------------------------------
| DELETE RELATIONSHIP u8-20-10 PROCESSES id-25000;   |
|----------------------------------------------------|
| DEL REL u8-20-20 PROCESSES id-25000;               |
|----------------------------------------------------|
| DEL REL u8-20-30-10 PROCESSES id-25000;            |
 ----------------------------------------------------
```

By using the relationship-selection-clause in the DELETE RELATIONSHIP command, we could have specified the relationship removal in one step:

```
 ----------------------------------------------------
| DEL REL                                            |
|   SELECT ALL RELATIONSHIPS FOR  id-25000;          |
 ----------------------------------------------------
```

In any case, having deleted its relationships, we can then remove the entity itself:

```
 ----------------------------------------------------
| DELETE ENTITY id-25000;                            |
 ----------------------------------------------------
```

2.4.2 Changing the Names of an Entity.

It´s sometimes necessary to change an entity´s access or descriptive name. For example, to change the access name od-25000 to rpt-25000, we could say:

```
 ----------------------------------------------------
| MODIFY ACCESS-NAME  od-25000                       |
|   TO  rpt-25000;                                   |
 ----------------------------------------------------
```

3. THE DICTIONARY OUTPUT FACILITY

The three IRDS output commands have very general formats; each can be used for applications ranging from the ad-hoc querying of a dictionary to the generation of highly structured, written reports. These commands, GENERAL OUTPUT, OUTPUT IMPACT OF CHANGE, and OUTPUT SYNTAX have the same overall structure, which we can represent as

```
command-imperative
  output-selection
  output-formatting
  output-routing;
```

Output-selection is the selection of the precise list of dictionary entities that will comprise the output. The syntax for this selection is the same for all three commands, and is discussed in Section 3.1.

Output-formatting is the specification of the precise information to be displayed, and the format of the display. It includes entity sort information, discussed in Section 3.2, and the "show" options. These differ for the three output commands, and will be illustrated in the individual discussions of each command.

The optional output-routing is simply

```
 -----------------------------------------------------
| ROUTE TO destination , destination ...              |
 -----------------------------------------------------
```

where destination is implementor defined.

3.1 OUTPUT SELECTION

The output-selection clauses allow the user to select the list of entities to be output. This section will illustrate one selection mode, namely the inclusion of the selection criteria within the output command itself. A user can also specify entity-lists and procedures; these techniques will be discussed in Chapter 6.

The user specifies selection criteria by first identifying the overall category of entities desired (entity selection), and then narrowing this list using combinations of restriction criteria. Thus, our command representation becomes

```
command-imperative
  SELECT entity-selection WHERE entity-restriction
  output-formating
  output-routing;
```

3.1.1 Entity Selection.

There are four alternatives for the initial entity selection:

(1) The collection of all entities accessible to a given user can be specified by

```
 -----------------------------------------------------
| SELECT ALL ENTITIES                                 |
 -----------------------------------------------------
```

(2) Selection can be made according to the strings of characters representing access names. For example, to select all entities whose access names begin with "u8-", followed by any single character, followed by "0", we could specify

```
 -----------------------------------------------------
| SELECT ENTITIES WITH ACCESS-NAME = u8-?0            |
 -----------------------------------------------------
```

(3) Selection can be made according to the strings of characters representing descriptive names. To select all entities whose descriptive names contain the string "Budget" and end with the string "SM", we could specify

```
 -----------------------------------------------------
| SELECT ENTITIES WITH DESCRIPTIVE-NAME = *Budget*SM  |
 -----------------------------------------------------
```

(4) Finally, entities can be selected according to how they are related to a specified entity. For example, to specify all entities that are related to (i.e., members of at least one relationship whose other member is) either rd-25091 or rd-25311, we could say

```
 -----------------------------------------------------
| SELECT ENTITIES DIRECTLY RELATED TO rd-25091,       |
|                                     rd-25311        |
 -----------------------------------------------------
```

To specify all entities that are contained, directly or indirectly, in u8, we can say

```
 -----------------------------------------------------
| SELECT ENTITIES RELATED TO u8 VIA CONTAINS          |
 -----------------------------------------------------
```

3.1.2 Entity Restriction.

A typical entity-restriction is composed of a boolean combination of restriction clauses. Some illustrations are:

```
 -----------------------------------------------------
| ENTITY-TYPE = FILE, RECORD, ELEMENT                 |
 -----------------------------------------------------
```

which specifies a restriction to entities of one of three entity-types;

```
 -----------------------------------------------------
| NO RELATIONSHIPS EXIST                              |
 -----------------------------------------------------
```

which identifies "orphan" entities;

```
 -----------------------------------------------------
| SECURITY = "datamgr"                                |
 -----------------------------------------------------
```

and

```
 -----------------------------------------------------
| DATE-ADDED >= "830609000000"                        |
 -----------------------------------------------------
```

which restrict to entities with these attributes;

```
 -----------------------------------------------------
| DESCRIPTION = "*database*"                          |
 -----------------------------------------------------
```

which finds entities whose descriptions contain the string "database"; and

```
 ---------------------------------------------------------
| DESCRIPTIVE-NAME´LENGTH >= 32                           |
 ---------------------------------------------------------
```

and

```
 ---------------------------------------------------------
| DESCRIPTION´LINES >= 10                                 |
 ---------------------------------------------------------
```

which test for entities whose descriptive name length and number of description lines satisfy the given criteria.

3.1.3 Full Output Selection Examples.

Putting some of these clauses together, we can have

```
 ---------------------------------------------------------
| SELECT ALL ENTITIES WHERE  DESCRIPTION = "*database*"|
 ---------------------------------------------------------

 ---------------------------------------------------------
| SELECT ALL ENTITIES WHERE                               |
|   ENTITY-TYPE = FILE AND DESCRIPTION = "*database*"     |
 ---------------------------------------------------------

 ---------------------------------------------------------
| SELECT ENTITIES WITH ACCESS-NAME = u8-?0                |
|   WHERE NO RELATIONSHIPS EXIST AND                      |
|   (DATE-ADDED >= "830609000000"                         |
|      OR SECURITY = "datamgr")                           |
 ---------------------------------------------------------
```

3.2 SORTING THE ENTITIES

Since the output commands produce basically a list of selected entities and associated data, it´s often important to specify a specific sort sequence for the entities.

If no sort-clause is specified, the entities are displayed in the order they are retrieved. If we want to sort by entity-type, within entity-type by assigned access name, and for entities with a given assigned access name by the DATE-CREATED attribute, the sort clause would be

```
   SORT
     SEQUENCE = ENTITY-TYPE, ASSIGNED ACCESS-NAME,
       DATE-ADDED
```

For any sort parameter, we can specify ascending or descending order. Thus,

```
   SORT  SEQUENCE = ENTITY-TYPE, ASSIGNED ACCESS-NAME,
     (DATE-ADDED  DESCENDING)
```

would list entities of the same type, with the same assigned access name, in reverse chronological order.

3.3 THE GENERAL OUTPUT COMMAND

3.3.1 SHOWing All Information.

All information about selected entities can be displayed using the SHOW ALL option. Therefore, to generate what amounts to a dump or catalog of the dictionary contents, we can

```
 OUTPUT DICTIONARY
   SELECT ALL ENTITIES
   SHOW ALL;
```

3.3.2 Names and Types.

To limit the display to the access or descriptive names of the entities, we would say

```
   SHOW ACCESS-NAME
```

or

```
   SHOW DESCRIPTIVE-NAME
```

respectively.

If we´re only interested in finding out the entity-type of each selected entity, we would say

```
SHOW ENTITY-TYPE
```

3.3.3 Attributes of Entities.

The SHOW ALL clause automatically displays all attributes of each selected entity. If we´re not using SHOW ALL, we can still display all attributes by including

```
SHOW ALL ATTRIBUTES
```

Likewise, we can specify that no attributes are to be displayed:

```
SHOW NO ATTRB
```

just certain attributes:

```
SHOW ATTRB SECURITY, FREQUENCY,
  DESCRIPTION (1 THROUGH 5)
```

or all attributes except certain ones:

```
SHOW ALL ATTRB EXCEPT DESCRIPTION
```

3.3.4 Relationships of Entities.

The amount of information that can be output concerning the relationships of the selected entities is highly variable. All relationship information can be displayed by

```
SHOW ALL RELATIONSHIPS
```

perhaps limited as to direction, as in

```
 ------------------------------------------------------
|   SHOW ALL FORWARD RELS                              |
 ------------------------------------------------------
```

or

```
 ------------------------------------------------------
|   SHOW ALL INVERSE RELS                              |
 ------------------------------------------------------
```

We can narrow the display of relationships either by explicit inclusion

```
 ------------------------------------------------------
|   SHOW RELS RECORD-CONTAINS-ELEMENT                  |
|             RECORD-CONTAINED-IN-FILE                 |
 ------------------------------------------------------
```

or by explicit exclusion

```
 ------------------------------------------------------
|   SHOW ALL RELS EXCEPT CONTAINS, PROGRAM-CALLS-MODULE|
 ------------------------------------------------------
```

All attributes of relationships will be displayed, unless we explicitly suppress them, as in

```
 ------------------------------------------------------
|   SHOW RELS CONTAINS AND NO ATTRIBUTES               |
 ------------------------------------------------------
```

3.3.5 Output Counts.

Finally, we can use the SHOW clause to provide various counts of the displayed information, as with

```
 ------------------------------------------------------
|   SHOW ALL ATTRIBUTES                                |
|   SHOW ALL RELS                                      |
|   SHOW ENTITIES´COUNT, ATTRIBUTES´COUNT              |
|     RELATIONSHIPS´COUNT                              |
 ------------------------------------------------------
```

3.3.6 Complete Command Examples.

Putting together the various combinations, we have such examples of the GENERAL OUTPUT command as

```
OUTPUT DICTIONARY
  SELECT ALL ENTITIES WHERE
    DESCRIPTION = "*security*" OR
    DESCRIPTION = "*password*"
  SORT SEQUENCE = ENTITY-TYPE, ACCESS-NAME
  SHOW ACCESS-NAME
  SHOW ATTRB DESCRIPTION;
```

and

```
OUTPUT DICTIONARY
  SELECT ENTITIES ACCESS-NAME = id-25000, od-25000
  SHOW "DOCUMENT REPORT" ON FIRST PAGE
  SHOW RELS PROCESSED-BY
  SHOW RELS´COUNT;
```

Note that the last example specifies a report title.

3.4 THE OUTPUT IMPACT-OF-CHANGE COMMAND

A simple, "find all" application of this command might ask for a list of all entities affected by a change to entity u8. This could be specified as

```
OUTPUT IMPACT
  SELECT ENTITIES ACCESS-NAME = u8;
```

More complex select clauses can be specified as for the GENERAL OUTPUT command. Note that the absence of a SHOW clause implies the display of just the access names of the impacted entities.

If we wanted to include a title, we could include something like

```
 ------------------------------------------------------
|   SHOW "A CHANGE TO SYSTEM u8 WOULD AFFECT THE       |
|     FOLLOWING ENTITIES:"                             |
 ------------------------------------------------------
```

If we wanted to restrict the list to impacted FILE, RECORD, and ELEMENT entities, say, we would specify

```
 ------------------------------------------------------
|   SHOW ONLY FILE, RECORD, ELEMENT                    |
 ------------------------------------------------------
```

To specify the display of the descriptive names of the impacted entities, and for these entities, the name of the person who added the entity to the dictionary, we could say

```
 ------------------------------------------------------
|   SHOW DESCRIPTIVE-NAME                              |
|   SHOW ATTRIBUTE ADDED-BY                            |
 ------------------------------------------------------
```

Thus, a more realistic example of this command would be:

```
 ------------------------------------------------------
| OUTPUT CUMULATIVE IMPACT                             |
|   SELECT ENTITIES WITH ACCESS-NAME = u8-?0           |
|     WHERE ENTITY-TYPE = SYSTEM                       |
|   SORT SEQUENCE = (ACCESS-NAME ASCENDING)            |
|   SHOW ATTRB LAST-MODIFIED-BY;                       |
 ------------------------------------------------------
```

As applied to our example dictionary, this command would generate the display of a single list of those SYSTEM entities that would be affected by a change to any of the entities u8-20, u8-30, and u8-40. Thus, the output would be the list of entities

```
  u8 u8-20-10, u8-20-20, u8-20-30, u8-30-10, u8-30-20,
  u8-30-30, u8-30-40
```

where, for each of these output entities, the name of the person who last modified that entity is also displayed.

3.5 THE OUTPUT SYNTAX COMMAND

The OUTPUT SYNTAX command is basically a simplified version of the GENERAL OUTPUT command that displays its output in the form of a sequence of BEGIN ENTITY and BEGIN RELATIONSHIP pseudo-commands. Each pseudo-command produced in this way is syntactically consistent with the corresponding ADD ENTITY or ADD RELATIONSHIP command. Thus, there is little need in the OUTPUT SYNTAX command for additional formatting; the principal function of the SHOW clause is to specify the relationships that are to be displayed.

An example of the command is:

```
OUTPUT SYNTAX
  SELECT ALL ENTITIES WHERE
    ENTITY-TYPE = DOCUMENT
  SORT SEQUENCE = (ACCESS-NAME ASCENDING)
  SHOW ALL RELATIONSHIPS AND NO ATTRIBUTES;
```

Applied to our example dictionary, this command would produce a display something like:

```
BEGIN ENTITY id-25000 ENTITY-TYPE = DOCUMENT
  DESCRIPTIVE-NAME = ASCAD-Table-Change-Request
  WITH ATTRIBUTES
    ADDED-BY = "John-Smith",
    DATE-ADDED = "840331194053",
          . . .
    SECURITY = "datamgr";

BEGIN u8-20-10 PROCESSES id-25000;

BEGIN u8-20-20 PROCESSES id-25000;

BEGIN u8-20-30-10 PROCESSES id-25000;

BEGIN ENTITY od-25000 ENTITY-TYPE = DOCUMENT
          . . .
    SECURITY = datamgr;

BEGIN u8-40 PROCESSES od-25000;
```

4. CUSTOMIZING THE DICTIONARY SCHEMA

The objects and their interrelationships specified in the Core IRDS System-Standard Schema may not precisely match the requirements of a given organization. The documented properties of certain real-world entities that are to be modeled may not match anything in the System-Standard Schema, desirable relationship-types may not be present, etc. Therefore, the IRDS allows an organization to fully customize the System-Standard Schema. This feature, called "extensibility," permits the definition of new entity-types, relationship-types, attribute-types, and other schema objects. The Command Language itself is not modifiable in the Core IRDS.

4.1 CHANGING THE NAME OF A META-ENTITY

Perhaps the simplest application of extensibility is for an organization to change the name of an entity-type, attribute-type, or other meta-entity.

The ASCAD dictionary refers to PROGRAMs as "operations." If we want to accommodate this usage, we could very easily rename PROGRAM by specifying

```
MODIFY META-ENTITY-NAME FROM
  PROGRAM TO OPERATION;
```

4.2 CHANGING AN EXISTING ENTITY-TYPE

Although the collection of entity-types provided by the System-Standard Schema will probably be adequate for most applications at most organizations, the specific characteristics of these entity-types will often require customization. We will first show how we would create a new attribute-type and associate it with a given entity-type. Then we will illustrate the modification of an entity-type´s meta-attributes.

4.2.1 Assigning a New Attribute-Type.

When we defined the FILE entities in our continuing example, we applied to them only DESCRIPTION, SECURITY, IDENTIFICATION-NAMES, and NUMBER-OF-RECORDS attributes. Suppose it were important to record, and then select entities based upon, the storage medium (tape, disk, etc.) or the retention (temporary, permanent) of the FILEs. We could accomplish this by defining the attribute-types MEDIUM and RETENTION, and then associating them with the FILE entity-type.

An attribute-type is an example of a meta-entity. Therefore, we define the two new attribute-types by:

```
 ------------------------------------------------------
| ADD META-ENTITY MEDIUM                               |
|   META-ENTITY-TYPE = ATTRIBUTE-TYPE;                 |
|------------------------------------------------------|
| ADD META-ENTITY RETENTION                            |
|   META-ENTITY-TYPE = ATTRIBUTE-TYPE;                 |
 ------------------------------------------------------
```

We now associate these attribute-types with the entity-type FILE. That is, we inform the IRDS that MEDIUM and RETENTION are to be allowable attribute-types for FILEs. This association is done by establishing meta-relationships between the meta-entity FILE and each of the meta-entities MEDIUM and RETENTION:

```
 ------------------------------------------------------
| ADD META-RELATIONSHIP                                |
|   FROM FILE TO MEDIUM                                |
|   WITH META-ATTRIBUTES                               |
|     SINGULAR/PLURAL = SINGULAR;                      |
|------------------------------------------------------|
| ADD META-RELATIONSHIP                                |
|   FROM FILE TO RETENTION                             |
|   WITH SING/PL = SINGULAR;                           |
 ------------------------------------------------------
```

Note the use of the meta-attribute SINGULAR/PLURAL on the meta-relationship to specify that only one MEDIUM attribute and one RETENTION attribute can be assigned to a given FILE. Note also that the establishment of these two meta-relationships automatically "installs" the respective new meta-entities.

If we later decide to undo this change with respect to, say, the RETENTION attribute-type, we would first remove the meta-relationship, then remove the meta-entity itself:

```
-------------------------------------------------------
| DELETE META-RELATIONSHIP                             |
|   FROM FILE TO RETENTION;                            |
|------------------------------------------------------|
| DELETE META-ENTITY RETENTION;                        |
-------------------------------------------------------
```

4.2.2 Modifying a Meta-Attribute.

The table in Section 9.7 of the Core IRDS Specifications lists the meta-attribute-types associated with each meta-entity-type. A given entity-type (a meta-entity of type entity-type) such as FILE or PROGRAM has associated with it a number of meta-attribute-types such as ADDED-TO-SCHEMA-BY, ALTERNATE-META-ENTITY-NAME, and MAXIMUM-NAME-LENGTH. The *values* of these meta-attribute-types (the meta-attributes) then define the characteristics of the entity-type. We notice from the table that the meta-attribute-type MAXIMUM-NAME-LENGTH is optional for entity-types. Suppose we wanted to ensure that all ELEMENTs had assigned access names of length no greater than 16 characters (while allowing their descriptive names to be of arbitrary length). What we would need to do is to modify the characteristics of the meta-entity ELEMENT by changing the value of MAXIMUM-NAME-LENGTH:

```
-------------------------------------------------------
| MODIFY META-ENTITY ELEMENT                           |
|   WITH META-ATTRIBUTES                               |
|     MAXIMUM-NAME-LENGTH = 16;                        |
-------------------------------------------------------
```

The IRDS will check the contents of the dictionary to make sure that no existing ELEMENT has an assigned access name longer than 16 characters.

4.3 CREATING A NEW ENTITY-TYPE

To fully define a new entity-type in the schema, we need to perform the following steps:

1. We create the new entity-type, by adding to the schema an entity-type meta-entity.

(At this point, the new entity-type has associated with it only those attribute-types, such as DATE-ADDED and COMMENTS, that are common to all entity-types (the meta-attribute-type COMMON has a value of "yes".) We must explicitly associate with the entity-type any additional attribute-types.)

2. We construct the set of relationship-types that the new entity-type is to be a member of, by adding to the schema the corresponding set of relationship-type meta-entities.

3. We assign (if appropriate) each new relationship-type to its relationship-class-type, by adding a meta-relationship between the relationship-type and the relationship-class-type.

(At this point the two prospective members of each relationship-type (the new entity-type and an existing entity-type) will not have been explicitly connected within that relationship-type.)

4. We assign to each of the new relationship-types the new entity-type and an existing entity-type (as the two members of the relationship-type), by specifying a set of meta-relationships.

5. We create, if necessary, any new attribute-types that the new entity-type or any of the new relationship-types might need, by adding to the schema the corresponding meta-entities.

6. We link the appropriate attribute-types to the new entity-type and to the new relationship-types, by specifying a set of meta-relationships.

7. We install in the schema the new entity-type and the new relationship-types.

We will illustrate all this with a straightforward creation of a "DRAWING" entity-type. To simplify the example, we will assume that DRAWING is a member of only the "DOCUMENT-CONTAINS-DRAWING" relationship-type.

4.3.1 Creating the Meta-Entity.

```
ADD META-ENTITY DRAWING
  META-ENTITY-TYPE = ENTITY-TYPE
  WITH META-ATTRIBUTES
    ALTERNATE-META-ENTITY-NAME = DRW
    PURPOSE =
      "A DRAWING ENTITY REPRESENTS A"
      "COLLECTION OF GRAPHIC AND NON GRAPHIC"
      "(ALPHANUMERIC) INFORMATION";
```

4.3.2 Defining the Relationship-Types.

By our assumptions, we need to define only the DOCUMENT-CONTAINS-DRAWING relationship-type:

```
ADD META-ENTITY DOCUMENT-CONTAINS-DRAWING
  META-ENTITY-TYPE = RELATIONSHIP-TYPE
  WITH
    INVERSE-NAME = DRAWING-CONTAINED-DOCUMENT
    ALTERNATE-META-ENTITY-NAME = DOC-CON-DRW;
```

4.3.3 Specifying the Relationship-Class-Type.

This section and the next highlight the important fact that the character string comprising an IRDS name has no inherent meaning. Simply naming a new relationship-type "DOCUMENT-CONTAINS-DRAWING" does not cause the IRDS to infer that "DOCUMENT", "DRAWING", or "CONTAINS" are in any way associated with the relationship-type. We could have named the relationship-type "xxxxxx", as long as we perform the next two operations.

Here, we tell the IRDS that DOCUMENT-CONTAINS-DRAWING is a "CONTAINS" relationship-type:

```
ADD META-RELATIONSHIP
  FROM DOCUMENT-CONTAINS-DRAWING TO CONTAINS;
```

4.3.4 Assigning Members to the Relationship-Type.

We need to tell the IRDS that both the new entity-type (DRAWING) and the existing entity-type (DOCUMENT) are members of the relationship-type DOCUMENT-CONTAINS-DRAWING, and indicate the relative positions of the two entity-types within the relationship-type:

```
ADD META-RELATIONSHIP
  FROM DOCUMENT-CONTAINS-DRAWING TO DOCUMENT
  POSITION = 1;
----------------------------------------------------
ADD META-RELATIONSHIP
  FROM DOCUMENT-CONTAINS-DRAWING TO DRAWING
  POS = 2;
```

4.3.5 Creating New Attribute-Types.

A DRAWING entity would no doubt need to have available a collection of attribute-types not in the System-Standard Schema to describe its lines, shading, color, labels, etc. We define here only one such attribute-type, COLOR:

```
ADD META-ENTITY COLOR
  META-ENTITY-TYPE = ATTRIBUTE-TYPE;
```

4.3.6 Associating the Appropriate Attribute-Types.

We must explicitly associate with DRAWING all non-common attribute-types, such as COLOR. For example:

```
ADD META-RELATIONSHIP
  FROM DRAWING TO COLOR
  WITH META-ATTRIBUTES
    SING/PL = PLURAL;
```

4.3.7 Installing the Schema Descriptors.

We install DRAWING, then DOCUMENT-CONTAINS-DRAWING:

```
 ---------------------------------------------------------
| INSTALL                                                 |
|   DRAWING, DOCUMENT-CONTAINS-DRAWING;                   |
 ---------------------------------------------------------
```

The new entity-type DRAWING and the new relationship-type DOCUMENT-CONTAINS-DRAWING are now fully defined, and we can begin to create corresponding entities and relationships in the dictionary.

4.4 THE SCHEMA OUTPUT FACILITY

The SCHEMA OUTPUT command, which selects and displays the schema metadata, has a structure very similar to that of the dictionary output commands. We specify:

```
OUTPUT SCHEMA
  SELECT meta-entity-selection WHERE meta-entity-restriction
  output-formatting
  output-routing;
```

4.4.1 Meta-Entity-Selection.

We can select for output either all meta-entities

```
 ---------------------------------------------------------
|   SELECT ALL                                            |
 ---------------------------------------------------------
```

or an explicit list of them:

```
 ---------------------------------------------------------
|   SELECT FILE, RECORD, FILE-CONTAINS-RECORD,            |
|          DATE-ADDED                                     |
 ---------------------------------------------------------
```

4.4.2 Meta-Entity-Restriction.

A typical meta-entity restriction expression is composed of a boolean combination of restriction clauses. For example:

```
META-ENTITY-TYPE = ENTITY-TYPE, RELATIONSHIP-TYPE
```

restricts the output to those meta-entities that are either entity-types or relationship-types, and

```
(MINIMUM-NAME-LENGTH >= 12) AND
  (MAXIMUM-NAME-LENGTH <= 24)
```

narrows the selection to those entity-types whose instances have assigned access names that are specified to be strings of between 12 and 24 characters.

4.4.3 Full Selection Example.

To select those entity-types that were either entered into the schema through extensibility or modified since January 1, 1983, we specify:

```
SELECT ALL WHERE
  META-ENTITY-TYPE = ENTITY-TYPE AND
  (LEVEL = EXTENDED OR
      DATE-MODIFIED >= 830101000000)
```

4.4.4 Sorting the Meta-Entities.

We can sort by meta-entity-type and/or by meta-attributes. For example:

```
SORT SEQUENCE =
  LEVEL, META-ENTITY-TYPE,
    (DATE-MODIFIED DESCENDING)
```

4.4.5 Output Formatting.

For each meta-entity selected, we can specify the display of all associated schema information: the meta-attributes of the meta-entities, the meta-relationships involving the meta-entities, and the set of all meta-entities that are meta-related to the given meta-entities.

Thus we might have:

```
SHOW "EVERYTHING"
SHOW ALL
```

```
SHOW ALL META-ATTRIBUTES
```

```
SHOW META-ATTRIBUTES DATE-ADDED, ADDED-BY
```

```
SHOW META-ATTRIBUTES COMMENTS
SHOW META-RELATIONSHIPS
```

```
SHOW ALL META-ATTRIBUTES
SHOW DIRECTLY RELATED META-ENTITIES
SHOW INDIRECTLY RELATED META-ENTITIES WHERE
  META-ENTITY-TYPE = RELATIONSHIP-CLASS-TYPE
```

4.4.6 A Complete Example.

```
OUTPUT SCHEMA
  SELECT ALL WHERE
    LEVEL = EXTENDED AND
      META-ENTITY-TYPE = ENTITY-TYPE,
        RELATIONSHIP-TYPE, ATTRIBUTE-TYPE
  SORT SEQUENCE = META-ENTITY-TYPE
  SHOW ALL META-ATTRIBUTES
  SHOW META-RELATIONSHIPS;
```

This command would, if applied to the schema after the commands in Section 4.3, display information on the new meta-entities created in that section. That is, this command would display DRAWING and DOCUMENT-CONTAINS-DRAWING, with their respective meta-attributes and meta-relationships.

5. IRDS NAMING AND CONTROL FACILITIES

5.1 THE VERSIONING FACILITY

5.1.1 Defining Versions.

The Core IRDS has no commands that deal exclusively with the Versioning Facility. A user specifies a variation identifier for a new entity in the ADD ENTITY command, and constructs variations of existing entities with the MODIFY ENTITY, COPY ENTITY, or MODIFY ENTITY LIFE-CYCLE-PHASE commands.

For example, an entity representing a Spanish language version of the DOCUMENT id-25000 could be created using:

```
COPY ENTITY  id-25000  WITH RELATIONSHIPS
  TO NEW VERSION = (Spanish);
```

Unless later modified, id-25000(Spanish) would have the same attributes as does id-25000 (except for such audit attributes as the value of DATE-ADDED) and would participate in relationships with the same entities. Since id-25000 has a descriptive name, the IRDS would assign one to the new entity. This descriptive name, ASCAD-Table-Change-Request(Spanish), would be formed from the assigned descriptive name of the original entity and the version-identifier of the new entity.

The IRDS assigns an implicit revision-number of 1 to a newly added entity or entity variation; a user can explicitly assign a revision-number when using the MODIFY ENTITY, COPY ENTITY, and MODIFY ENTITY LIFE-CYCLE-PHASE commands. If, in these commands, the user specifies NEW VERSION with no variation identifier, the IRDS automatically increments the revision-number for the newly created entity.

For example, we could represent a revision to the Spanish language DOCUMENT by:

```
 ----------------------------------------------------------
| MODIFY ENTITY  id-25000(Spanish)                         |
|   NEW VERSION;                                           |
 ----------------------------------------------------------
```

which creates the new entity id-25000(Spanish:2).

5.1.2 Using Versions.

Version identifiers can figure in entity selection criteria.

Both

```
 ----------------------------------------------------------
| SELECT ENTITIES WITH ACCESS-NAME = *(Operation*)         |
 ----------------------------------------------------------
```

and

```
 ----------------------------------------------------------
| SELECT ALL ENTITIES WHERE VARIATION = Operation          |
 ----------------------------------------------------------
```

specify the selection of all entities with a variation-name of "Operation".

Likewise,

```
 ----------------------------------------------------------
| SELECT ALL ENTITIES WHERE                                |
|   ENTITY-TYPE = ELEMENT AND REVISION = LOWEST            |
 ----------------------------------------------------------
```

will find the earliest revision of each ELEMENT.

5.2 LIFE-CYCLE-PHASES

5.2.1 Defining New Phases.

A life-cycle-phase is a meta-entity in the schema. The Core System-Standard Schema provides four life-cycle-phases, UNCONTROLLED-PHASE, CONTROLLED-PHASE, ARCHIVED-PHASE, and SECURITY-PHASE. If we want to define a new phase, we use the ADD META-ENTITY command:

```
ADD META-ENTITY TEST-PHASE
  META-ENTITY-TYPE = LIFE-CYCLE-PHASE
  WITH ALT-MNAME = TEST;
```

The IRDS will assign TEST-PHASE to the life-cycle-phase class UNCONTROLLED. TEST-PHASE will automatically be installed in the schema.

5.2.2 Placing Entities in Phases.

As explained in Section 5.4, when an entity is created in the dictionary, it is automatically assigned to the life-cycle-phase associated with the view currently in effect. If a user wants to transfer existing entities from one phase to another, the MODIFY ENTITY LIFE-CYCLE-PHASE command is used. Thus, to transfer the entities id-25000 and od-25000 from UNCONTROLLED-PHASE to TEST-PHASE:

```
MODIFY ENTITY LIFE-CYCLE-PHASE
  FOR id-25000, od-25000
  FROM UNCONTROLLED-PHASE TO TEST-PHASE;
```

Since NEW VERSION wasn´t specified, this command will "move" the two entities. If NEW VERSION had been used, then two additional entities with the same assigned access names (but with new versions) as the originals would be created.

Such transfers must obey the life-cycle-phase integrity rules.

5.2.3 Using Life-Cycle-Phases.

Life-cycle-phases can figure in entity selection, sorting, and display criteria.

We can restrict selected entities to be in specific life-cycle-phases. Thus:

```
  SELECT ALL ENTITIES WHERE
    LIFE-CYCLE-PHASE = TEST-PHASE
```

```
   SELECT ALL ENTITIES WHERE
     (PHASE <= CONTROLLED-PHASE) AND
       (PHASE <= ARCHIVED-PHASE)
```

The latter SELECT clause specifies all entities in all UN-CONTROLLED phases. This kind of restriction can be used in all dictionary output commands.

We can sort according to life-cycle-phase:

```
   SORT SEQUENCE = LIFE-CYCLE-PHASE
```

The following clause is used in dictionary output commands to specify the display of the life-cycle-phase of each selected entity:

```
   SHOW LIFE-CYCLE-PHASE
```

5.3 QUALITY-INDICATORS

The Quality-Indicator Facility in the IRDS allows an organization to arbitrarily define quality-indicator descriptors and assign them to entities. These descriptors are then available for documentation and search purposes.

5.3.1 Defining Quality-Indicators.

The Core System-Standard Schema does not include any quality-indicators, so an organization will have to explicitly define a set of them to make use of this capability. Since a quality-indicator is a meta-entity, new indicators are created by the ADD META-ENTITY command:

```
 ADD META-ENTITY PROPOSED-INDICATOR
   META-ENTITY-TYPE = QUALITY-INDICATOR
   WITH ALT-MNAME = PROPOSED;
```

5.3.2 Assigning Quality-Indicators to Entities.

When an entity is created or modified using the ADD ENTITY, MODIFY ENTITY, or COPY ENTITY commands, the user can assign a quality-indicator to that entity. For example:

```
-----------------------------------------------------
| ADD ENTITY  fd-62000  ETYPE = FILE                  |
|   QUALITY = PROPOSED;                               |
-----------------------------------------------------
-----------------------------------------------------
| COPY ENTITY  rd-25091 WITH RELS TO  rd-65091        |
|   QUALITY = APPROVED;                               |
-----------------------------------------------------
```

Before the execution of these two examples, quality-indicators named PROPOSED and APPROVED must have been explicitly added to the schema.

5.3.3 Using Quality-Indicators.

Quality-indicators can figure in entity selection and display criteria.

We can restrict selected entities to those assigned a given quality-indicator. For example:

```
-----------------------------------------------------
|   SELECT ALL ENTITIES WHERE                         |
|     QUALITY = APPROVED                              |
-----------------------------------------------------
```

The quality-indicator is one of the characteristics of an entity that can be displayed by a GENERAL OUTPUT or an OUTPUT IMPACT-OF-CHANGE command. To do this, we simply include the clause:

```
-----------------------------------------------------
|   SHOW QUALITY                                      |
-----------------------------------------------------
```

5.4 VIEWS

5.4.1 Defining VIEWs.

Structurally, a VIEW is an entity in the dictionary created and manipulated much as any other entity. The subset of the dictionary defined by the VIEW is specified by ENTITY-TYPE-NAME and EXCLUDE-RELATIONSHIPS attributes that are assigned to the VIEW. Note that these attribute-types are part of the DICTIONARY-PERMISSIONS attribute-group-type, which is also used to specify access permissions associated with the VIEW.

For example, we can construct a VIEW allowing full access to all SYSTEM entities in life-cycle-phase TEST-PHASE, and to all relationships involving these SYSTEMs except for those of type SYSTEM-GOES-TO-SYSTEM:

```
ADD ENTITY Sys-View
  ENTITY-TYPE = VIEW
  WITH ATTRIBUTES
    DICTIONARY-PERMISSIONS =
      (ENTITY-TYPE-NAME = SYSTEM,
       EXCLUDE-RELATIONSHIPS = SYSTEM-GOES-TO-SYSTEM,
       READ-PERMISSION = YES,
       ADD-PERMISSION = YES,
       MODIFY-PERMISSION = YES,
       DELETE-PERMISSION = YES,
       MODIFY-PHASE-PERMISSION = YES,
       ADMINISTRATOR-PERMISSION = YES),
    LIFE-CYCLE-PHASE-NAME = Test-Phase;
```

We will illustrate the use of the other attribute-types within DICTIONARY-PERMISSIONS more fully in Section 5.5.

5.4.2 Placing Entities in VIEWs.

When a VIEW is created, entities of the types specified by ENTITY-TYPE-NAME attributes may already exist in the given life-cycle-phase. In this case the VIEW would immediately contain these entities.

An entity or relationship created in the dictionary will automatically be placed in the effective VIEW, and will simultaneously become visible in all other VIEWs (within the appropriate phase) that specify the entity-type. Likewise, an entity transferred from one phase to another using the MODIFY ENTITY LIFE-CYCLE-PHASE command will become visible through all appropriate VIEWs in the new phase.

5.4.3 Using Views.

Dictionary output commands and the BUILD ENTITY-LIST command each allow the user to override, for the execution of that command, the effective VIEW. Thus, if we say:

```
OUTPUT DICTIONARY
  USING VIEW = Sys-View
  SELECT ALL ENTITIES WHERE
    DESCRIPTION = "*security*" OR
    DESCRIPTION = "*password*"
  SORT SEQUENCE = ENTITY-TYPE, ACCESS-NAME
  SHOW ACCESS-NAME
  SHOW ATTRB DESCRIPTION;
```

the output will contain only those entities and relationships (specified by the SELECT clause) that are visible through Sys-View, (i.e., only SYSTEM entities, and relationships involving SYSTEMs except for SYSTEM-GOES-TO-SYSTEM.)

If we had said

```
  USING VIEW = ALL
```

the output would contain all the selected entities and relationships visible through <u>any</u> VIEW associated with the IRDS user.

5.5 CORE SECURITY

The Core IRDS Security Facility allows an organization to restrict access to the schema and the dictionary. This is done by

1. Defining appropriate VIEWs of the dictionary.

2. Constructing a DICTIONARY-USER entity for each user.

3. Relating the two kinds of entities by DICTIONARY-USER-HAS-VIEW relationships.

The specific access permissions and restrictions are attributes on the VIEW and DICTIONARY-USER entities.

5.5.1 The Use of Views.

A VIEW entity has attributes that allow it to specify the degree of access permitted to users who access the dictionary through the VIEW. The attribute-types comprising the attribute-group-type DICTIONARY-PERMISSION include those that can grant or withhold permission to read, add to, modify, delete from, and modify the life-cycle-phases of the entities and relationships visible through the VIEW.

Thus, we can define a VIEW that includes all RECORDs and ELEMENTs, and allows these RECORDs and ELEMENTs to be read and added to, but not modified in any way:

```
ADD ENTITY R-E-View
  ENTITY-TYPE = VIEW
  WITH ATTRIBUTES
    DICTIONARY-PERMISSIONS =
     (ENTITY-TYPE-NAME = RECORD,
      ENTITY-TYPE-NAME = ELEMENT,
      READ-PERMISSION = YES,
      ADD-PERMISSION = YES,
      MODIFY-PERMISSION = NO,
      DELETE-PERMISSION = NO,
      MODIFY-PHASE-PERMISSION = NO,
      ADMINISTRATOR-PERMISSION = NO),
    LIFE-CYCLE-PHASE-NAME = Production-Phase;
```

5.5.2 The Dictionary-User Entity.

A DICTIONARY-USER entity is created and maintained (by the "dictionary administrator") for each user of the IRDS. Such an entity has attributes that allow or forbid:

* The use of the IRDS Command Language (COMMAND-LANGUAGE-PERMISSION = "yes" or "no").

* The ability to change assigned access or descriptive names (RENAME-PERMISSION = "yes" or "no").

* Various levels of access to the schema (details given in the discussion of the SCHEMA-PERMISSION-1, ..., SCHEMA-PERMISSION-5 attribute-types in Section 10.2.4 of the Core Specifications)

Thus, we can define for user John Doe a corresponding DICTIONARY-USER entity that allows him the use of the Command Language, and the ability to obtain output on, but not modify, the dictionary schema:

```
ADD ENTITY  John-Doe
  ENTITY-TYPE = DICTIONARY-USER
  WITH ATTRIBUTES
    COMMAND-LANGUAGE-PERMISSION = YES,
    RENAME-PERMISSION = NO,
    SCHEMA-PERMISSION-1 = NO,
    SCHEMA-PERMISSION-2 = NO,
    SCHEMA-PERMISSION-3 = YES,
    SCHEMA-PERMISSION-4 = YES,
    SCHEMA-PERMISSION-5 = YES;
```

5.5.3 Assigning Views to Dictionary Users.

Using DICTIONARY-USER-HAS-VIEW relationships, a IRDS user can be assigned as many VIEWs as necessary to customize his or her access privileges. The IRDS user´s default VIEW is also assigned at this time:

```
ADD RELATIONSHIP
  John-Doe DICTIONARY-USER-HAS-VIEW Sys-View
  WITH ATTRIBUTES
    DEFAULT-VIEW = YES;
```

6. DICTIONARY ENTITY-LISTS AND PROCEDURES

6.1 ENTITY-LISTS

An IRDS user can avoid re-specifying entity selection criteria by creating and later using an entity-list.

6.1.1 Creating Entity-Lists.

The BUILD ENTITY-LIST command takes the same entity selection criteria clause used in the dictionary output commands and creates a stored, re-usable list of entities. Thus,

```
BUILD ENTITY-LIST
  SELECT ENTITIES WITH
    ACCESS-NAME = *ASCAD*
  WHERE
    (DESCRIPTION = "*database*" AND
      NO RELATIONSHIPS EXIST) OR
    (REVISION < HIGHEST AND PHASE = TEST)
  LIST-NAME = DB-old-list;
```

If we had left out the last line in this command, the collection of selected entities would have become the current list.

6.1.2 Manipulating Entity-Lists.

New entity-lists can be created from existing lists by the use of appropriate set operations.

By specifying

```
UNION DB-Old-List-1, DB-Old-List-2 = DB-List;
```

we form DB-List, which is the list of all unique entities in DB-Old-List-1 and DB-Old-List-2. If we had specified a null-mark instead of DB-Old-List-2, DB-List would have been the union of DB-Old-List-1 and the current list.

```
 ------------------------------------------------------
| INTERSECT ASCAD-1, ASCAD-2, ASCAD-3, ASCAD-4;        |
 ------------------------------------------------------
```

assigns to the current list those entities that are in each of the lists ASCAD-1, ASCAD-2, ASCAD-3, and ASCAD-4.

By specifying

```
 ------------------------------------------------------
| DIFFERENCE Budget-Recs, Account-Recs                 |
|   = New-Recs;                                        |
 ------------------------------------------------------
```

we form New-Recs, the list of entities that are either in Budget-Recs but not in Account-Recs, or in Account-Recs but not in Budget-Recs.

```
 ------------------------------------------------------
| SUBTRACT Total-List, NA-List = Back-List;            |
 ------------------------------------------------------
```

assigns to Back-List the set of entities that are in Total-List but not in NA-List.

6.1.3 Using Entity-Lists.

We can use previously defined entity-lists in the DELETE ENTITY and DELETE RELATIONSHIP commands, and in dictionary output commands.

To delete the entities specified in DB-Old-List, we say simply:

```
 ------------------------------------------------------
| DELETE ENTITY                                        |
|   USING ENTITY-LIST = DB-Old-List;                   |
 ------------------------------------------------------
```

In output commands, a reference to an existing entity-list would replace the explicit SELECT ... WHERE ... criteria. For example, we can use DB-Old-List in an OUTPUT IMPACT-OF-CHANGE command as follows:

```
 ------------------------------------------------------
| OUTPUT IMPACT                                        |
|   USING ELIST = DB-Old-List;                         |
 ------------------------------------------------------
```

In the last example, we could have used SORT or SHOW clauses, if we had wished.

6.1.4 Entity-List Utilities.

The current entity-list can, at any time, be given an explicit name. For example:

```
NAME CURRENT ENTITY-LIST Hold-Progs;
```

The command:

```
OUTPUT ENTITY-LIST
  LIST-NAME = List-3
  SHOW "LIST OF REQUIRED ELEMENTS";
```

outputs the contents of List-3, along with the specified title.

If we issue the command

```
OUTPUT ENTITY-LIST NAMES;
```

we will be provided with the names of all defined entity-lists.

6.2 PROCEDURES

The IRDS provides facilities for two kinds of procedures: output procedures, which contain the syntax of output commands; and entity-list-procedures. The latter are necessary because entity-lists created during a given user session are saved only until the completion of that session. A longer retention time is undesirable because the content of the dictionary is dynamic, and a rigid entity-list can easily become obsolete and lose its utility. Instead, we store the _procedure_ for creating the entity-list. The procedure is much more likely to remain valid, and can be run at any time to re-create the (perhaps updated) entity-list.

6.2.1 Creating Procedures.

Procedures can be created "on the fly" within other commands, or separately in a save procedure command.

We can specify within any dictionary output command that the command syntax be saved as an output procedure. For example, we can modify the example in Section 3.3.6:

```
OUTPUT DICTIONARY
  SELECT ALL ENTITIES WHERE
    DESCRIPTION = "*security*" OR
    DESCRIPTION = "*password*"
  SORT SEQUENCE = ENTITY-TYPE, ACCESS-NAME
  SHOW ACCESS-NAME
  SHOW ATTRB DESCRIPTION
  PROCEDURE-NAME = P-Sec
  PROCEDURE-DESCRIPTION = "Procedure to output names
    and descriptions of security related entities.";
```

We have used here the optional procedure description clause. P-Sec will refer to this entire GENERAL OUTPUT command.

We can use the same syntax within the BUILD ENTITY-LIST command to create an entity-list procedure. Thus, in the example in Section 6.1.1, if we had said

```
BUILD ENTITY-LIST
  SELECT ENTITIES WITH
    ACCESS-NAME = *ASCAD*
  WHERE
    (DESCRIPTION = "*database*" AND
      NO RELATIONSHIPS EXIST) OR
    (REVISION < HIGHEST AND PHASE = TEST)
  PROCEDURE-NAME = DB-Old-Proc;
```

we would have stored DB-Old-Proc, the procedure to generate the given entity-list, rather than DB-Old-List, the entity-list itself. (Actually, an entity-list is always created by the BUILD ENTITY-LIST command, in this case it would be the current list.)

If we issue a dictionary output or BUILD ENTITY-LIST command without saving the syntax in a procedure, we can do a retroactive save by using a SAVE OUTPUT PROCEDURE or SAVE ENTITY-LIST PROCEDURE command, as appropriate. Thus,

```
 ----------------------------------------------------
| SAVE OUTPUT PROCEDURE-NAME P-Sec;                  |
 ----------------------------------------------------
```

will store, as P-Sec, the syntax of the last dictionary output command to have been executed, and

```
 ----------------------------------------------------
| SAVE ENTITY-LIST PROCEDURE                         |
|   LIST-NAME = DB-Old-List  P-NAME = DB-Old-Proc;   |
 ----------------------------------------------------
```

will store, as DB-Old-Proc, the syntax that had been used to produce DB-Old-List, (i.e., the "SELECT ... WHERE ..." clause).

6.2.2 Using Procedures.

Entity-list procedures can be used either within other commands or explicitly, using a RUN ENTITY-LIST PROCEDURE command; output procedures are executed using the RUN OUTPUT PROCEDURE command.

Entity-list procedures can be used analogously to and in the same contexts as entity-lists themselves--in DELETE ENTITY and DELETE RELATIONSHIP commands, and in dictionary output commands. Thus:

```
 ----------------------------------------------------
| DELETE ENTITY                                      |
|   USING PROCEDURE = DB-Old-Proc;                   |
 ----------------------------------------------------
```

would, if executed during the same user session, have the same effect as the DELETE example in Section 6.1.3. Unlike the entity-list, however, the procedure would be available indefinitely, unless deleted or redefined.

We can execute P-Sec by specifying

```
 ----------------------------------------------------
| RUN OUTPUT PROCEDURE P-Sec;                        |
 ----------------------------------------------------
```

Likewise, we can execute an entity-list procedure to create an entity-list:

```
RUN ENTITY-LIST PROCEDURE  DB-Old-Proc
  LIST-NAME = DB-Old-List;
```

6.2.3 Procedure Utilities.

We can delete a procedure by specifying, for example:

```
DELETE ELIST PROCEDURE DB-Old-Proc;
```

or

```
DELETE OUTPUT PROC P-Sec;
```

The syntax of one or more procedures can be displayed. For example:

```
OUTPUT PROCEDURE SYNTAX
  SELECT P-Sec, DB-Old-Proc
  SHOW "OUR PROCEDURES";
```

A listing of the names of our procedures can be obtained by issuing:

```
OUTPUT PROCEDURE-NAMES
  SHOW PDESC;
```

This command will display the names and descriptions associated with our procedures.

7. THE IRD TO IRD INTERFACE

Since the commands for the IRD to IRD Interface Facility contain implementor defined clauses, we can illustrate syntax in the following examples only to the level of these clauses.

7.1 EXPORTING TO AN EMPTY DICTIONARY

We will transport the example application (the source) to an arbitrary target environment, thus creating a copy of the entire dictionary. The target environment could be at another organization, perhaps one that uses a different IRDS.

We first use the BUILD ENTITY-LIST command to specify the subset of the source dictionary (in this case, all of it) to be exported:

```
BUILD ENTITY-LIST
  SELECT ALL ENTITIES
  LIST-NAME = All-Example;
```

We then export the source schema and dictionary contents into files whose names are appropriate for the source dictionary's operating environment:

```
EXPORT DICTIONARY
  USING ENTITY-LIST = All-Example
  SCHEMA EXPORT FILE = <schema-export-file-name>
  DICTIONARY EXPORT FILE = <dictionary-export-file
                                             -name>;
```

We now move to the target environment and create an empty dictionary using the Core System-Standard Schema:

```
CREATE DICTIONARY  <new-dictionary-name>
  SCHEMA IS STANDARD;
```

One effect of such a CREATE DICTIONARY command is the establishment of an UNCONTROLLED life-cycle-phase, whose name is implementor dependent. Let´s assume that the name is LOAD-PHASE.

In the generalized export/import procedure, the next step is to check the compatibility of source and target schemas. Since we are now in the target environment, we specify:

```
 -----------------------------------------------------
| CHECK SCHEMA                                        |
|   SOURCE SCHEMA IS IN FILE <schema-export-file-name>;|
 -----------------------------------------------------
```

Since no changes were made to the Core System-Standard Schema during the construction of the example, we should receive at this point an implementor defined message confirming compatibility. We can then import the source schema and dictionary into life-cycle-phase LOAD-PHASE of the target dictionary:

```
 -----------------------------------------------------
| IMPORT DICTIONARY                                   |
|   SCHEMA EXPORT FILE = <schema-export-file-name>    |
|   DICTIONARY EXPORT FILE = <dictionary-export-file  |
|                                              -name> |
|   LIFE-CYCLE-PHASE = Load-Phase;                    |
 -----------------------------------------------------
```

7.2 EXPORTING TO AN EXISTING DICTIONARY

Suppose we want to add the contents of our source dictionary to a non-empty dictionary in the target environment. After exporting the source schema and dictionary, we would compare the source and target schemas. The CHECK SCHEMA command would tell us about any incompatibilities; we would then modify the source or target schemas as required. If we modify the source schema, we then need to re-issue the above EXPORT DICTIONARY command. We then conclude with the IMPORT DICTIONARY command.

8. MISCELLANEOUS TOPICS IN THE CORE

8.1 SETTING SESSION DEFAULTS

We can use the SET command to set the session defaults for effective VIEW, check vs. execute mode, and attribute coding vs. decoding. Thus:

```
SET
  VIEW = DIV-101
  MODE = CHECK
  SHOW ATTRIBUTES ENCODED
  SAVE;
```

specifies the effective VIEW, and sets the mode and code settings to the opposite of the normal defaults. The SAVE clause goes further, and specifies these settings to be our future defaults.

8.2 DISPLAYING SESSION RELATED INFORMATION

The command

```
STATUS ALL;
```

displays all status information.

We can also find out the status of a particular option. Like the previous example, the following displays all status information, but does so by asking about each option individually:

```
STATUS
  DICTIONARY
  ENTITY-LIST
  MODE
  VIEWS
  PROFILES
```

```
|   DEFAULTS;                                                   |
 ---------------------------------------------------------------
```

8.3 OBTAINING HELP

In an interactive session, we can obtain on-line help from the IRDS.

```
 ---------------------------------------------------------------
| HELP;                                                         |
 ---------------------------------------------------------------
```

or

```
 ---------------------------------------------------------------
| HELP ALL;                                                     |
 ---------------------------------------------------------------
```

displays the names of the commands that we are authorized to use. We can then ask for more information on one of these:

```
 ---------------------------------------------------------------
| HELP DELETE-RELATIONSHIP;                                     |
 ---------------------------------------------------------------
```

We can also ask the system to explain an error or warning message:

```
 ---------------------------------------------------------------
| HELP MESSAGE;                                                 |
 ---------------------------------------------------------------

 ---------------------------------------------------------------
| HELP MESSAGE <message-identifier>;                            |
 ---------------------------------------------------------------
```

The first of these explains any error messages encountered in the execution of the previous command. The second, using the implementor defined <message-identifier>, generates an explanation of that particular error message.

9. IRDS MODULES

9.1 ENTITY LEVEL SECURITY

The Entity Level Security optional module allows an organization to restrict users of the IRDS from accessing individual entities in the dictionary. This is done by associating an ACCESS-CONTROLLER to each entity for which protection is desired. A user attempting to access a protected entity would then need to use a VIEW with access keys that match the access locks on the controller.

9.1.1 Securing Entities.

Entities can be secured at the time they are created with the ADD ENTITY, MODIFY ENTITY, or COPY ENTITY commands. In addition, existing entities can be secured using ADD SECURITY. These commands automatically create the necessary ACCESS-CONTROLLER entities and the "secured-by" relationships.

Suppose that we want to add the secured entity fd-30210. We could say:

```
ADD ENTITY  fd-30210  E-TYPE = FILE
  ASSIGN SECURITY
    NEW CONTROLLER = Division-Controller;
```

Among the results of this command are the creation by the IRDS of the ACCESS-CONTROLLER entity Division-Controller, and the relationship fd-30210 FILE-SECURED-BY-ACCESS-CONTROLLER Division-Controller. The IRDS will generate and assign to Division-Controller a read lock and a write lock.

Similarly, we could have:

```
COPY ENTITY  fd-30210 TO fd-30211
  ASSIGN SECURITY
    CONTROLLER = Division-Controller;
```

and

```
MODIFY ENTITY  fd-30210
  NEW VERSION  INCLUDE SECURITY;
```

In the last two examples, the new entity is associated with the existing controller Division-Controller via the appropriate FILE-SECURED-BY-ACCESS-CONTROLLER relationship.

Suppose we want to secure the existing entities rd-25310, rd-25345, and dd-02200. We could say:

```
ADD SECURITY
  TO  rd-25310, rd-25345, dd-02200
  NEW CONTROLLER = Code-Controller;
```

Having secured the desired entities, we now make them available through the appropriate VIEWs. We do this by assigning to the VIEWs the read or write access keys that will match the locks on the relevant controllers. Assume that rd-25310, rd-25345, and dd-02200 are each visible through the VIEW Table-View. Then we can grant permission to read the three entities to anyone who has access to Table-View by:

```
ADD READ ACCESS-KEY
  FROM CONTROLLERS = Code-Controller
  TO VIEWS = Table-View;
```

Likewise for write permission.

9.1.2 Changing the Security of Entities.

Code-Controller, the controller associated with rd-25310, rd-25345, and dd-02200, can be replaced by Division-Controller:

```
MODIFY SECURITY
  TO  rd-25310, rd-25345, dd-02200
    FROM CONTROLLER = Code-Controller
    TO CONTROLLER = Division-Controller;
```

To delete entity level security entirely from these entities, we can say:

```
DELETE SECURITY
  ON rd-25310, rd-25345, dd-02200
  CONTROLLER = Division-Controller;
```

We can delete the access keys (the read key, in this case) from Table-View:

```
DELETE ACCESS-KEY
  FROM CONTROLLER = Code-Controller
  TO VIEWS = Table-View;
```

9.2 APPLICATION PROGRAM (CALL) INTERFACE

The format of a "CALL" statement using this interface depends completely on the language in which the application program is written and on implementor defined parameters. Therefore, no Command Language examples for this module can be provided.

9.3 SUPPORT OF STANDARD DATA MODELS

This optional module specifies no modifications of or additions to the Command Language of the Core IRDS. The module does contain a collection of new entity-, relationship-, and attribute-types that allow the documentation and modeling of network and relational database environments. Thus, for example, we can create and manipulate DATABASE, SCHEMA, and SET entities, and associated relationships and attributes, as specified in the module.

APPENDIX A: COMPLETE LISTING OF EXAMPLE CREATION

Appendix A is a complete listing of all the IRDS commands illustrated or alluded to in Section 2.3. As such, it represents the "official" definition of the application dictionary described in Section 2.1 and referred to throughout this document.

```
------------------------------------------------------------
CREATE DICTIONARY Example
  SCHEMA IS STANDARD;
------------------------------------------------------------
ADD ENTITY u8
  ENTITY-TYPE = SYSTEM
  DESCRIPTIVE-NAME =
    ASCAD-Database-Information-System
  WITH ATTRIBUTES
    DESCRIPTION =
        "This system provides the necessary tools
      for maintaining the Air Staff Codes and
      Descriptions (ASCAD) Database. The ASCAD
      Database contains all common (corporate) data
      elements which are codes and their respective
      descriptions. The tools provide the
      capability:
        1. To control access to the database
          a. single record at a time
          b. groups of records
        2. To update the tables in the database
        3. To produce reports from the database
        4. To create tapes containing database
           information
        5. To display information online.",
    SECURITY = "datamgr";
------------------------------------------------------------
ADD ENTITY  u8-20  ENTITY-TYPE = SYSTEM
  DESCRIPTIVE-NAME = ASCAD-Database-Update
  WITH ATTRIBUTES
    DESCRIPTION  (START = 100  INCREMENT = 10)
    =
       "This subsystem provides the capability for
      the Air Staff to update the contents of the
      ASCAD Database.",
    SYSTEM-CATEGORY = "subsystem",
    SECURITY = "datamgr";
------------------------------------------------------------
ADD ENTITY  u8-30  ENTITY-TYPE = SYSTEM
  DESCRIPTIVE-NAME = ASCAD-GCOS-File-Creation
  WITH ATTRIBUTES
```

```
    DESCRIPTION =
        "This subsystem provides the capability to
      create change transactions in a format
      compatible with the GCOS batch system.",
    SYSTEM-CATEGORY = "subsystem",
    SECURITY = "datamgr";
-------------------------------------------------------
ADD ENTITY  u8-40  ENTITY-TYPE = SYSTEM
  DESCRIPTIVE-NAME = ASCAD-Table-Report
  WITH ATTRIBUTES
    DESCRIPTION =
        "This subsystem provides the capability to
      create a report for each table in the ASCAD
      database. The report includes all data elements
      and all records in the table.",
    SYSTEM-CATEGORY = "subsystem",
    SECURITY = "datamgr";
-------------------------------------------------------
ADD RELATIONSHIP
  u8  SYSTEM-CONTAINS-SYSTEM  u8-20;
-------------------------------------------------------
ADD RELATIONSHIP
  u8  SYSTEM-CONTAINS-SYSTEM  u8-30;
-------------------------------------------------------
ADD RELATIONSHIP
  u8  SYSTEM-CONTAINS-SYSTEM  u8-40;
-------------------------------------------------------
ADD ENTITY  u8-20-10  ENTITY-TYPE = SYSTEM
  DESCRIPTIVE-NAME = Initiate-ASCAD-Change-Request
  WITH ATTRIBUTES
    DESCRIPTION =
      "This procedure involves the manual operation
      of filling out the update request form and
      submitting it to the proper OPR.",
    SYSTEM-CATEGORY = "procedure",
    SECURITY = "datamgr";
-------------------------------------------------------
ADD ENTITY  u8-20-20  ENTITY-TYPE = SYSTEM
  DESCRIPTIVE-NAME = ASCAD-Table-Update-Input
  WITH ATTRIBUTES
    DESCRIPTION =
      "This procedure provides the online
      instructions necessary to activate the computer
      procedure to do the actual updating of the
      ASCAD tables.",
    SYSTEM-CATEGORY = "procedure",
    SECURITY = "datamgr";
-------------------------------------------------------
ADD ENTITY  u8-20-30  ENTITY-TYPE = SYSTEM
  DESCRIPTIVE-NAME = ASCAD-Table-Update
  WITH ATTRIBUTES
```

```
    DESCRIPTION =
      "This computer procedure receives the update
      modification requests from the user's response
      and changes them accordingly on the specified
      table of the ASCAD database.",
    SYSTEM-CATEGORY = "procedure",
    SECURITY = "datamgr";
------------------------------------------------------
ADD ENTITY  u8-30-10  ENTITY-TYPE = SYSTEM
  DESCRIPTIVE-NAME = Initiate-GCOS-Trans-File
  WITH ATTRIBUTES
    DESCRIPTION =
      "This administrative procedure outlines the
      steps that are required to initiate the ASCAD
      GCOS transaction file creation.",
    SYSTEM-CATEGORY = "procedure",
    SECURITY = "datamgr";
------------------------------------------------------
ADD ENTITY  u8-30-20  ENTITY-TYPE = SYSTEM
  DESCRIPTIVE-NAME = Produce-GCOS-Trans-File
  WITH ATTRIBUTES
    DESCRIPTION =
      "This interactive procedure provides the user
      with the capability to create a tape containing
      ASCAD table changes in a format compatible with
      the BPC System (DM changes) or the Data Codes
      Master (DCM) System (DCMF changes).",
    SYSTEM-CATEGORY = "procedure",
    SECURITY = "datamgr";
------------------------------------------------------
ADD ENTITY  u8-30-30  ENTITY-TYPE = SYSTEM
  DESCRIPTIVE-NAME = Create-GCOS-Trans-File
  WITH ATTRIBUTES
    DESCRIPTION =
      "This computer procedure provides the user
      with the capability to select the type of
      change tape (DM or DCMF) to create and to input
      the date/time of the last change transmitted
      to GCOS.",
    SYSTEM-CATEGORY = "procedure",
    SECURITY = "datamgr";
------------------------------------------------------
ADD ENTITY  u8-30-40  ENTITY-TYPE = SYSTEM
  DESCRIPTIVE-NAME = Verify-GCOS-Trans-File
  WITH ATTRIBUTES
    DESCRIPTION =
      "This procedure allows the user to verify the
      creation of the ASCAD GCOS transaction file.
      This is an online check to see if any problems
      have occurred while processing the absentee to
      create the GCOS tape.",
    SYSTEM-CATEGORY = "procedure",
    SECURITY = "datamgr";
-----------------------------------------------------
ADD RELATIONSHIP
u8-20 SYSTEM-CONTAINS-SYSTEM u8-20-10;
-----------------------------------------------------
ADD RELATIONSHIP
u8-20 SYSTEM-CONTAINS-SYSTEM u8-20-20;
-----------------------------------------------------
ADD RELATIONSHIP
u8-20 SYSTEM-CONTAINS-SYSTEM u8-20-30;
-----------------------------------------------------
ADD RELATIONSHIP
u8-30 SYSTEM-CONTAINS-SYSTEM u8-30-10;
-----------------------------------------------------
ADD RELATIONSHIP
u8-30 SYSTEM-CONTAINS-SYSTEM u8-30-20;
-----------------------------------------------------
ADD RELATIONSHIP
u8-30 SYSTEM-CONTAINS-SYSTEM u8-30-30;
-----------------------------------------------------
ADD RELATIONSHIP
u8-30 SYSTEM-CONTAINS-SYSTEM u8-30-40;
-----------------------------------------------------
ADD ENTITY  u8-20-30-10
  ENTITY-TYPE = PROGRAM
    DESCRIPTIVE-NAME = ASCAD-Update;
-----------------------------------------------------
ADD ENTITY  md-00772
  ENTITY-TYPE = MODULE
    DESCRIPTIVE-NAME = generalized-ASCAD-update;
-----------------------------------------------------
ADD ENTITY md-00771 ENTITY-TYPE = MODULE
  DESCRIPTIVE-NAME = generalized-mrds;
-----------------------------------------------------
ADD RELATIONSHIP
  u8-20-30-10 PROGRAM-CALLS-MODULE md-00772;
-----------------------------------------------------
ADD RELATIONSHIP
  md-00772 MODULE-CALLS-MODULE md-00771;
-----------------------------------------------------
ADD RELATIONSHIP
  u8-20-30 SYSTEM-CONTAINS-PROGRAM u8-20-30-10;
-----------------------------------------------------
ADD RELATIONSHIP
  u8-20-10 SYSTEM-GOES-TO-SYSTEM u8-20-20;
-----------------------------------------------------
ADD RELATIONSHIP
  u8-20-20 SYSTEM-GOES-TO-SYSTEM u8-20-30;
-----------------------------------------------------
ADD RELATIONSHIP
```

```
  u8-20-30 SYSTEM-GOES-TO-SYSTEM u8-20-20;
-------------------------------------------------------
ADD RELATIONSHIP
  u8-30-20 SYSTEM-COMES-FROM-SYSTEM u8-30-10;
-------------------------------------------------------
ADD RELATIONSHIP
  u8-30-30 SYSTEM-COMES-FROM-SYSTEM u8-30-20;
-------------------------------------------------------
ADD RELATIONSHIP
  u8-30-20 SYSTEM-COMES-FROM-SYSTEM u8-30-30;
-------------------------------------------------------
ADD RELATIONSHIP
  u8-30-40 COMES-FROM u8-30-30;
-------------------------------------------------------
ADD RELATIONSHIP
  u8-30-30 CONTAINS NEW PROGRAM u8-30-30-10;
-------------------------------------------------------
ADD RELATIONSHIP
  u8-30-30 CONTAINS NEW PROGRAM u8-30-30-20;
-------------------------------------------------------
ADD RELATIONSHIP
  u8-30-30-10 GOES-TO u8-30-30-20;
-------------------------------------------------------
ADD ENTITY fd-05031  ENTITY-TYPE = FILE
  DESCRIPTIVE-NAME = Manpower-ASCAD-SM;
-------------------------------------------------------
ADD ENTITY fd-25091  ENTITY-TYPE = FILE
  DESCRIPTIVE-NAME = Countries/States-SM;
-------------------------------------------------------
ADD RELATIONSHIP  fd-05031 CONTAINS fd-25091;
-------------------------------------------------------
ADD ENTITY fd-05007  ENTITY-TYPE = FILE
  DESCRIPTIVE-NAME = ASCAD-Budget-Tables-SM;
-------------------------------------------------------
ADD ENTITY fd-00103 ENTITY-TYPE = FILE
  DESCRIPTIVE-NAME = ASCAD-Data-Model;
-------------------------------------------------------
ADD ENTITY fd-25093  ENTITY-TYPE = FILE
  DESCRIPTIVE-NAME = Commands-SM;
-------------------------------------------------------
ADD RELATIONSHIP fd-05007 CONTAINS fd-00103;
-------------------------------------------------------
ADD RELATIONSHIP fd-05007 CONTAINS fd-25093;
-------------------------------------------------------
ADD RELATIONSHIP fd-05007 CONTAINS fd-25091;
-------------------------------------------------------
ADD ENTITY rd-25091  ENTITY-TYPE = RECORD
  DESCRIPTIVE-NAME = Countries/States;
-------------------------------------------------------
ADD RELATIONSHIP
  fd-25091 FILE-CONTAINS-RECORD rd-25091;
```

```
COPY ENTITY
  rd-25091 WITH RELATIONSHIPS TO  rd-25311
  DESCRIPTIVE-NAME = Countries/States-NK;
-----------------------------------------------------------------
COPY ENTITY
  rd-25091 WITH RELS TO rd-25310
  DNAME = Countries/States-Key;
-----------------------------------------------------------------
COPY ENTITY
  rd-25091 WITH RELS TO rd-25345
  DNAME = Countries/States-Key-PR;
-----------------------------------------------------------------
ADD ENTITY dd-01093 ENTITY-TYPE = ELEMENT
  DESCRIPTIVE-NAME = County/State-Code;
-----------------------------------------------------------------
ADD ENTITY dd-01092 ENTITY-TYPE = ELEMENT
  DESCRIPTIVE-NAME = County/State-Abbreviation;
-----------------------------------------------------------------
ADD ENTITY dd-01333 ENTITY-TYPE = ELEMENT
  DESCRIPTIVE-NAME = Zoned-Interior-or-Overseas-Ind;
-----------------------------------------------------------------
ADD ENTITY dd-01325 ENTITY-TYPE = ELEMENT
  DESCRIPTIVE-NAME = US-and-Poss-or-Fgn-Cntry-Ind;
-----------------------------------------------------------------
ADD ENTITY dd-02075 ENTITY-TYPE = ELEMENT
  DESCRIPTIVE-NAME = Geographical-Region-World;
-----------------------------------------------------------------
ADD ENTITY dd-01021 ENTITY-TYPE = ELEMENT
  DESCRIPTIVE-NAME = Area-of-the-World;
-----------------------------------------------------------------
ADD RELATIONSHIP rd-25091 CONTAINS dd-01093;
-----------------------------------------------------------------
ADD RELATIONSHIP rd-25091 CONTAINS dd-01092;
-----------------------------------------------------------------
ADD RELATIONSHIP rd-25091 CONTAINS dd-01333;
-----------------------------------------------------------------
ADD RELATIONSHIP rd-25091 CONTAINS dd-01325;
-----------------------------------------------------------------
ADD RELATIONSHIP rd-25091 CONTAINS dd-02075;
-----------------------------------------------------------------
ADD RELATIONSHIP rd-25091 CONTAINS dd-01021;
-----------------------------------------------------------------
ADD RELATIONSHIP rd-25311 CONTAINS dd-01092;
-----------------------------------------------------------------
ADD RELATIONSHIP rd-25311 CONTAINS dd-01333;
-----------------------------------------------------------------
ADD RELATIONSHIP rd-25311 CONTAINS dd-01325;
-----------------------------------------------------------------
ADD RELATIONSHIP rd-25311 CONTAINS dd-02075;
ADD RELATIONSHIP rd-25311 CONTAINS dd-01021;
---------------------------------------------------------
ADD ENTITY dd-02200
  ETYPE = ELE  DNAME = Action-Code;
---------------------------------------------------------
ADD REL rd-25310 CONTAINS dd-02200;
---------------------------------------------------------
ADD REL rd-25345 CONTAINS dd-02200;
---------------------------------------------------------
ADD ENTITY id-25000  ENTITY-TYPE = DOCUMENT
  DESCRIPTIVE-NAME = ASCAD-Table-Change-Request;
---------------------------------------------------------
ADD ENTITY od-25000  ETYPE = DOC
DNAME = ASCAD-Table;
---------------------------------------------------------
ADD RELATIONSHIP
  u8 SYSTEM-PROCESSES-FILE fd-05031;
---------------------------------------------------------
ADD REL u8-40 SYSTEM-PROCESSES-FILE fd-05007;
---------------------------------------------------------
ADD REL u8-20-30-10 PROCESSES fd-05007;
---------------------------------------------------------
ADD REL u8-20-10 PROCESSES id-25000;
---------------------------------------------------------
ADD REL u8-20-20 PROCESSES id-25000;
---------------------------------------------------------
ADD REL u8-20-30-10 PROCESSES id-25000;
---------------------------------------------------------
ADD REL od-25000 PROCESSED-BY u8-40;
---------------------------------------------------------
MODIFY ENTITY u8-20-30-10
  WITH ATTRIBUTES
    DESCRIPTION =
      "Through the use of the mrds-database
      -supervisor the ASCAD database tables are
      updated as needed. One can add, change,
      delete, or display online any element of any
      ASCAD table.",
    SECURITY = "datamgr";
---------------------------------------------------------
MOD ENTITY md-00772
  WITH ATTRIBUTES
    DESCRIPTION =
      "This module provides a generalized means of
      updating tables in the ASCAD database and then
      recording the transaction on an audit trail.",
    SECURITY = "gks/dbm";
---------------------------------------------------------
MOD ENTITY md-00771
  WITH ATTRIBUTES
    DESCRIPTION =
      "This module provides a generalized means for
      manipulating data stored in an mrds relation.",
    SECURITY = "gks/dr";
--------------------------------------------------------
MOD ENTITY fd-05031
  WITH ATTRIBUTES
    DESCRIPTION =
      "This data submodel contains those tables in
      the ASCAD database used by Air Force Manpower
      (AF/MPM) systems and programs.",
    IDENTIFICATION-NAMES =
      (ALTERNATE-NAME = "data-codes-master",
       ALTERNATE-NAME-CONTEXT = "pll"),
    SECURITY = "datamgr";
--------------------------------------------------------
MOD ENTITY fd-25091
  WITH ATTRIBUTES
    DESCRIPTION =
      "This file (table) contains all valid location
      codes and their descriptive titles.",
    SECURITY = "datamgr",
    IDENTIFICATION-NAMES =
      (ALTERNATE-NAME = "cntry-states",
       ALTERNATE-NAME-CONTEXT = "pll"),
    NUMBER-OF-RECORDS = 293;
--------------------------------------------------------
MOD ENTITY fd-05007
  WITH ATTRIBUTES
    DESCRIPTION =
      "This file identifies all the tables existing
      in the ASCAD Budget submodel.",
    SECURITY = "datamgr",
    IDENTIFICATION-NAMES =
      (ALTERNATE-NAME = "ascad-bud-tables",
       ALTERNATE-NAME-CONTEXT = "pll"),
    NUMBER-OF-RECORDS = 50;
--------------------------------------------------------
MOD ENTITY fd-00103
  WITH ATTRIBUTES
    DESCRIPTION =
      "This file defines all the relations existing
      in the ASCAD data model.",
    SECURITY = "datamgr",
    IDENTIFICATION-NAMES =
      (ALTERNATE-NAME = "ascad-data-model",
       ALTERNATE-NAME-CONTEXT = "pll"),
    NUMBER-OF-RECORDS = 50;
--------------------------------------------------------
MOD ENTITY fd-25093
  WITH ATTRIBUTES
    DESCRIPTION =
      "This file (table) contains all valid major
      command codes and their descriptive titles.",
    SECURITY = "datamgr",
    IDENTIFICATION-NAMES =
      (ALTERNATE-NAME = "commands",
       ALTERNATE-NAME-CONTEXT = "pll"),
    NUMBER-OF-RECORDS = 56;
-----------------------------------------------------
MOD ENTITY rd-25091
  WITH ATTRIBUTES
    DESCRIPTION =
      "This record describes all the data elements
      contained in the location table.",
    IDENTIFICATION-NAMES =
      (ALTERNATE-NAME = "cntry-states",
       ALTERNATE-NAME-CONTEXT = "pll"),
    SECURITY = "datamgr";
-----------------------------------------------------
MOD ENTITY rd-25311
  WITH ATTRIBUTES
    DESCRIPTION =
      "This record identifies the non key fields.",
    IDENTIFICATION-NAMES =
      (ALTERNATE-NAME = "cntry-states",
       ALTERNATE-NAME-CONTEXT = "pll"),
    SECURITY = "datamgr";
-----------------------------------------------------
MOD ENTITY rd-25310
  WITH ATTRIBUTES
    DESCRIPTION =
      "This record allows for the entry of action
      codes and keys.",
    SECURITY = "datamgr";
-----------------------------------------------------
MOD ENTITY rd-25345
  WITH ATTRIBUTES
    DESCRIPTION =
      "This record allows for the entry of action
      code and keys.",
    IDENTIFICATION-NAMES =
      (ALTERNATE-NAME = "cntry-states",
       ALTERNATE-NAME-CONTEXT = "pll"),
    SECURITY = "datamgr";
-----------------------------------------------------
MOD ENTITY dd-01093
  WITH ATTRIBUTES
    DESCRIPTION =
      "A shared data field occupied by either
      country-code or state-code.",
    SECURITY = "datamgr",
    DATA-CLASS = "alphanumeric",
    IDENTIFICATION-NAMES =
      (ALTERNATE-NAME = "cntry-st-code",
       ALTERNATE-NAME-CONTEXT = "pll");
------------------------------------------------------
MOD ENTITY dd-01092
  WITH ATTRIBUTES
    DESCRIPTION =
      "A shared data field occupied by either
      Country or State.",
    SECURITY = "datamgr",
    DATA-CLASS = "alphabetic",
    IDENTIFICATION-NAMES =
      (ALTERNATE-NAME = "cntry-st-abbrv",
       ALTERNATE-NAME-CONTEXT = "pll");
------------------------------------------------------
MOD ENTITY dd-01333
  WITH ATTRIBUTES
    DESCRIPTION =
      "Indicates whether an installation is in the
      continental United States (Z1) or overseas
      (OS).",
    SECURITY = "datamgr",
    DATA-CLASS = "numeric",
    IDENTIFICATION-NAMES =
      (ALTERNATE-NAME = "zi-os-ind",
       ALTERNATE-NAME-CONTEXT = "pll");
------------------------------------------------------
MOD ENTITY dd-01325
  WITH ATTRIBUTES
    DESCRIPTION =
      "Indicates whether an installation is in the
      United States and its possessions or in a
      foreign country.",
    SECURITY = "datamgr",
    DATA-CLASS = "numeric",
    IDENTIFICATION-NAMES =
      (ALTERNATE-NAME = "usposs-for-ind",
       ALTERNATE-NAME-CONTEXT = "pll");
------------------------------------------------------
MOD ENTITY dd-02075
  WITH ATTRIBUTES
    DESCRIPTION =
      "This code is a geographical region
      representation of the world.",
    DATA-CLASS = "numeric",
    IDENTIFICATION-NAMES =
      (ALTERNATE-NAME = "region",
       ALTERNATE-NAME-CONTEXT = "pll");
------------------------------------------------------
MOD ENTITY dd-01021
  WITH ATTRIBUTES
```

```
    DESCRIPTION =
      "Represents a geographical boundary delineated
      in the unified command plan for personnel and
      manpower purposes.",
    SECURITY = "datamgr",
    DATA-CLASS = "alphanumeric",
    IDENTIFICATION-NAMES =
      (ALTERNATE-NAME = "area-world",
       ALTERNATE-NAME-CONTEXT = "pll");
-----------------------------------------------------
MODIFY RELATIONSHIP
  rd-25091 RECORD-CONTAINS-ELEMENT dd-01093
  WITH ATTRIBUTES
    RELATIVE-POSITION = 1;
-----------------------------------------------------
MODIFY RELATIONSHIP rd-25091 CONTAINS dd-01092
  WITH ATTRIBUTES
    RELATIVE-POSITION = 3;
-----------------------------------------------------
MOD REL rd-25091 CONTAINS dd-01033
  WITH ATTRIBUTES
    REL-POS = 8;
-----------------------------------------------------
MOD REL rd-25091 CONTAINS dd-01325
  WITH
    REL-POS = 9;
-----------------------------------------------------
MOD REL rd-25091 CONTAINS dd-02075
  WITH
    REL-POS = 10;
-----------------------------------------------------
MOD REL rd-25091 CONTAINS dd-01021
  WITH
    REL-POS = 11;
-----------------------------------------------------
MOD REL rd-25311 CONTAINS dd-01092
  WITH REL-POS = 1;
-----------------------------------------------------
MOD REL rd-25311 CONTAINS dd-01333
  WITH REL-POS = 6;
-----------------------------------------------------
MOD REL rd-25311 CONTAINS dd-01325
  WITH REL-POS = 7;
-----------------------------------------------------
MOD REL rd-25311 CONTAINS dd-02075
  WITH REL-POS = 8;
-----------------------------------------------------
MOD REL rd-25311 CONTAINS dd-01021
  WITH REL-POS = 9;
-----------------------------------------------------
MODIFY ENTITY dd-02200
```

```
  WITH ATTRIBUTES
    DESCRIPTION =
      "Indicates the action to be performed on a
      file in a database, add, change, delete, or
      print a record.",
    SECURITY = "datamgr",
    DATA-CLASS = "alphabetic",
    IDENTIFICATION-NAMES =
      (ALTERNATE-NAME = "action-code",
       ALTERNATE-NAME-CONTEXT = "pll");
-----------------------------------------------------
MODIFY RELATIONSHIP rd-25310 CONTAINS dd-02200
  WITH REL-POS = 1;
-----------------------------------------------------
MOD REL rd-25345 CONTAINS dd-02200
  WITH REL-POS = 1;
-----------------------------------------------------
MODIFY ENTITY id-25000
  WITH ATTRIBUTES
    DESCRIPTION =
      "This input form is used by the Air Staff
      Analyst to request a change to be made to a
      table in the ASCAD Database.",
    DOCUMENT-CATEGORY = "form",
    SECURITY = "datamgr";
-----------------------------------------------------
MOD ENTITY od-25000
  WITH ATTRIBUTES
    DESCRIPTION =
      "This output report displays all the
      contents of a table in the ASCAD Database.","
    DOCUMENT-CATEGORY = "report",
    SECURITY = "datamgr";
-----------------------------------------------------
MODIFY RELATIONSHIP u8 PROCESSES fd-05031
  WITH ACCESS-METHOD = "k";
-----------------------------------------------------
MOD REL u8-40 PROCESSES fd-05007
  WITH ATTRIBUTES
    ACCESS-METHOD = "k",
    FREQUENCY = "r";
-----------------------------------------------------
MOD REL u8-20-30-10 PROCESSES fd-05007
  WITH ACCESS-METHOD = "k";
-----------------------------------------------------
```

APPENDIX B: INDEX OF ALL COMMAND APPEARANCES

APPENDIX C: INDEX OF ALL CLAUSE APPEARANCES

REFERENCES

[1] ANSI X3H4, (draft proposed) American National Standard Information Resource Dictionary System: Part 1 -- Core Standard, ANSI TC X3H4/85-003, American National Standards Institute, New York, 1985.

[2] ANSI X3H4, (draft proposed) American National Standard Information Resource Dictionary System: Part 2 -- Entity-Level Security, ANSI TC X3H4/85-005, American National Standards Institute, New York, 1985.

[3] ANSI X3H4, (draft proposed) American National Standard Information Resource Dictionary System: Part 3 -- Application Program Interface, ANSI TC X3H4/85-006, American National Standards Institute, New York, 1985.

[4] ANSI X3H4, (draft proposed) American National Standard Information Resource Dictionary System: Part 4 -- Support of Standard Data Models, ANSI TC X3H4/85-007, American National Standards Institute, New York, 1985.

[5] Goldfine, A. H. and Konig, P. A., A Technical Overview of the Information Resource Dictionary System, NBSIR 85-3164, National Bureau of Standards, Gaithersburg, MD, 1985.

NBS-114A (REV. 2-80)

U.S. DEPT. OF COMM. BIBLIOGRAPHIC DATA SHEET (See instructions)	1. PUBLICATION OR REPORT NO. NBSIR-85/3165	2. Performing Organ. Report No.	3. Publication Date April 1985

4. TITLE AND SUBTITLE

Using the Information Resource Dictionary System Command Language

5. AUTHOR(S)

Alan H. Goldfine

6. PERFORMING ORGANIZATION (If joint or other than NBS, see instructions) NATIONAL BUREAU OF STANDARDS DEPARTMENT OF COMMERCE WASHINGTON, D.C. 20234	7. Contract/Grant No. 8. Type of Report & Period Covered Final

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS (Street, City, State, ZIP)

10. SUPPLEMENTARY NOTES

☐ Document describes a computer program; SF-185, FIPS Software Summary, is attached.

11. ABSTRACT (A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here)

This document introduces and provides examples of the Command Language of the draft proposed Information Resource Dictionary System (IRDS). A dictionary maintained by the U.S. Air Force is defined in the IRDS and used as a continuing example throughout the document. The dictionary is populated, manipulated; and reported on using the precise syntax of the Command Language. An appendix to the document provides a complete listing of the creation of the example. Other appendices provide indices of all command appearances and all clause appearances.

12. KEY WORDS (Six to twelve entries; alphabetical order; capitalize only proper names; and separate key words by semicolons)

command language; data dictionary; data dictionary system; data dictionary system standard; example book; Information Resource Dictionary System; IRDS.

13. AVAILABILITY [X] Unlimited ☐ For Official Distribution. Do Not Release to NTIS ☐ Order From Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402. [X] Order From National Technical Information Service (NTIS), Springfield, VA. 22161	14. NO. OF PRINTED PAGES 85 15. Price $11.50