

MICROCOPY RESOLUTION TEST CHART

2

AD-A181 953

DTIC FILE COPY

AFWAL-TR-86-4006  
Volume V  
Part 5



INTEGRATED INFORMATION  
SUPPORT SYSTEM (IISS)  
Volume V - Common Data Model Subsystem  
Part 5 - NDDL Processor Development Specification

General Electric Company  
Production Resources Consulting  
One River Road  
Schenectady, New York 12345

DTIC  
ELECTRONIC  
JUL 06 1987  
S D

Final Report for Period 22 September 1980 - 31 July 1985  
November 1985

Approved for public release; distribution is unlimited.

PREPARED FOR:

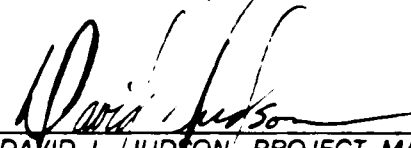
MATERIALS LABORATORY  
AIR FORCE WRIGHT AERONAUTICAL LABORATORIES  
AIR FORCE SYSTEMS COMMAND  
WRIGHT-PATTERSON AFB, OH 45433-6533

NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely related Government procurement operation, the United States Government thereby incurs no responsibility nor any obligation whatsoever, and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications or other data, is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.


This report has been reviewed by the Office of Public Affairs (ASD/PA) and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.

  
\_\_\_\_\_  
DAVID L. JUDSON, PROJECT MANAGER  
AFWAL/MLTC  
WRIGHT PATTERSON AFB OH 45433

5 Aug 1986  
\_\_\_\_\_  
DATE

FOR THE COMMANDER:

  
\_\_\_\_\_  
GERALD C. SHUMAKER, BRANCH CHIEF  
AFWAL/MLTC  
WRIGHT PATTERSON AFB OH 45433

7 Aug 86  
\_\_\_\_\_  
DATE

"If your address has changed, if you wish to be removed from our mailing list, or if the addressee is no longer employed by your organization please notify AFWAL/MLTC, W-PAFB, OH 45433 to help us maintain a current mailing list."

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

6a. ADDRESS (City, State and ZIP Code) <b>1 River Road Schenectady, NY 12345</b>		7a. ADDRESS (City, State and ZIP Code) <b>WPAFB, OH 45433-6533</b>	
8a. NAME OF FUNDING/SPONSORING ORGANIZATION <b>Materials Laboratory Air Force Systems Command, USAF</b>	8b. OFFICE SYMBOL (if applicable) <b>AFVAL/MLTC</b>	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER <b>F33615-80-C-5155</b>	
8c. ADDRESS (City, State and ZIP Code) <b>Wright-Patterson AFB, Ohio 45433</b>		10. SOURCE OF FUNDING NOS	
		PROGRAM ELEMENT NO. <b>78011F</b>	PROJECT NO. <b>7500</b>
		TASK NO. <b>62</b>	WORK UNIT NO. <b>01</b>
11. TITLE (Include Security Classification) (See Reverse)			
12. PERSONAL AUTHOR(S) <b>Althoff, J. L., Apicella, M. L., Bernier, M. P., Singh, S., Thompson, D. B., and Wong, J. P.</b>			
13a. TYPE OF REPORT <b>Final Technical Report</b>	13b. TIME COVERED <b>22 Sept 1980 - 31 July 1985</b>	14. DATE OF REPORT (Yr., Mo., Day) <b>1985 November</b>	15. PAGE COUNT <b>163</b>
16. SUPPLEMENTARY NOTATION <b>ICAM Project Priority 6201</b>			
The computer software contained herein are theoretical and/or references that in no way reflect Air Force-owned or -developed computer software.			
17. COSATI CODES		18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)	
FIELD	GROUP	SUB GR	
1508	0905		
19. ABSTRACT (Continue on reverse if necessary and identify by block number)			
<p>This document is the development specification establishing the functional requirements of the IISS Configuration Item Neutral Data Definition Language (NDDL) which is the primary tool used for maintaining the Common Data Model (CDM). <i>Keywords: IISS, Configuration Item Neutral Data Definition Language, Common Data Model</i></p>			
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED <input checked="" type="checkbox"/> SAME AS RPT. <input type="checkbox"/> DTIC USERS <input type="checkbox"/>		21. ABSTRACT SECURITY CLASSIFICATION <b>Unclassified</b>	
22a. NAME OF RESPONSIBLE INDIVIDUAL <b>David L. Judson</b>		22b. TELEPHONE NUMBER (Include Area Code) <b>513-255-8976</b>	22c. OFFICE SYMBOL <b>AFVAL/MLTC</b>

11. Title

Integrated Information Support System (IISS)  
Vol V - Common Data Model Subsystem  
Part 5 - NDDL Processor Development Specification



Accession For	
NTIS CRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification:	
By _____	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

PREFACE

This development specification covers the work performed under Air Force Contract F33615-80-C-5155 (ICAM Project 6201). This contract is sponsored by the Materials Laboratory, Air Force Systems Command, Wright-Patterson Air Force Base, Ohio. It was administered under the technical direction of Mr. Gerald C. Shumaker, ICAM Program Manager, Manufacturing Technology Division, through Project Manager, Mr. David Judson. The Prime Contractor was Production Resources Consulting of the General Electric Company, Schenectady, New York, under the direction of Mr. Alan Rubenstein. The General Electric Project Manager was Mr. Myron Hurlbut of Industrial Automation Systems Department, Albany, New York.

Certain work aimed at improving Test Bed Technology has been performed by other contracts with Project 6201 performing integrating functions. This work consisted of enhancements to Test Bed software and establishment and operation of Test Bed hardware and communications for developers and other users. Documentation relating to the Test Bed from all of these contractors and projects have been integrated under Project 6201 for publication and treatment as an integrated set of documents. The particular contributors to each document are noted on the Report Documentation Page (DD1473). A listing and description of the entire project documentation system and how they are related is contained in document FTR620100001, Project Overview.

The subcontractors and their contributing activities were as follows:

TASK 4.2

Subcontractors

Role

Boeing Military Aircraft  
Company (BMAC)

Reviewer

D. Appleton Company  
(DACOM)

Responsible for IDEF support,  
state-of-the-art literature  
search

General Dynamics/  
Ft. Worth

Responsible for factory view  
function and information  
models

Subcontractors

Role

Illinois Institute of  
Technology

Responsible for factory view  
function research (IITRI)  
and information models of  
small and medium-size business

North American Rockwell

Reviewer

Northrop Corporation

Responsible for factory view  
function and information  
models

Pritsker and Associates

Responsible for IDEF2 support

SofTech

Responsible for IDEFO support

TASKS 4.3 - 4.9 (TEST BED)

Subcontractors

Role

Boeing Military Aircraft  
Company (BMAC)

Responsible for consultation on  
applications of the technology  
and on IBM computer technology.

Computer Technology  
Associates (CTA)

Assisted in the areas of  
communications systems, system  
design and integration  
methodology, and design of the  
Network Transaction Manager.

Control Data Corporation  
(CDC)

Responsible for the Common Data  
Model (CDM) implementation and  
part of the CDM design (shared  
with DACOM).

D. Appleton Company  
(DACOM)

Responsible for the overall CDM  
Subsystem design integration  
and test plan, as well as part  
of the design of the CDM  
(shared with CDC). DACOM also  
developed the Integration  
Methodology and did the schema  
mappings for the Application  
Subsystems.



<u>Subcontractors</u>	<u>Role</u>
Digital Equipment Corporation (DEC)	Consulting and support of the performance testing and on DEC software and computer systems operation.
McDonnell Douglas Automation Company (McAuto)	Responsible for the support and enhancements to the Network Transaction Manager Subsystem during 1984/1985 period.
On-Line Software International (OSI)	Responsible for programming the Communications Subsystem on the IBM and for consulting on the IBM.
Rath and Strong Systems Products (RSSP) (In 1985 became McCormack & Dodge)	Responsible for assistance in the implementation and use of the MRP II package (PIOS) that they supplied.
SofTech, Inc.	Responsible for the design and implementation of the Network Transaction Manager (NTM) in 1981/1984 period.
Software Performance Engineering (SPE)	Responsible for directing the work on performance evaluation and analysis.
Structural Dynamics Research Corporation (SDRC)	Responsible for the User Interface and Virtual Terminal Interface Subsystems.

Prime contractors under other projects who have contributed to Test Bed Technology, their contributing activities and responsible projects are as follows:

<u>Contractors</u>	<u>ICAM Project</u>	<u>Contributing Activities</u>
Boeing Military Aircraft Company (BMAC)	1701, 2201, 2202	Enhancements for IBM node use. Technology Transfer to Integrated Sheet Metal Center (ISMC)

<u>Contractors</u>	<u>ICAM Project</u>	<u>Contributing Activities</u>
Control Data Corporation (CDC)	1502, 1701	IISS enhancements to Common Data Model Processor (CDMP)
D. Appleton Company (DACOM)	1502	IISS enhancements to Integration Methodology
General Electric	1502	Operation of the Test Bed and communications equipment.
Hughes Aircraft Company (HAC)	1701	Test Bed enhancements
Structural Dynamics Research Corporation (SDRC)	1502, 1701, 1703	IISS enhancements to User Interface/Virtual Terminal Interface (UI/VTI)
Systran	1502	Test Bed enhancements. Operation of Test Bed.

TABLE OF CONTENTS

		<u>Page</u>
SECTION 1.0	SCOPE .....	1-1
1.1	Identification .....	1-1
1.2	Functional Summary .....	1-1
SECTION 2.0	DOCUMENTS .....	2-1
2.1	Reference Documents .....	2-1
2.2	Terms and Abbreviations .....	2-2
SECTION 3.0	REQUIREMENTS .....	3-1
3.1	Computer Program Definition .....	3-1
3.1.1	System Capacities .....	3-1
3.1.2	Interface Requirements .....	3-1
3.1.2.1	Interface Block Diagram .....	3-2
3.1.2.2	Interface Requirements .....	3-3
3.2	Detailed Functional Requirements .....	3-3
3.2.1	Initialization .....	3-5
3.2.2	Input/Output .....	3-5
3.2.3	Error Handling .....	3-6
3.2.4	Parse Commands .....	3-7
3.2.5	Database Access .....	3-8
3.2.6	General Command Processing .....	3-10
3.2.7	Termination .....	3-12
3.2.8	Individual Command Processing .....	3-13
3.2.8.1	ALTER ALIAS .....	3-13
3.2.8.2	ALTER ATTRIBUTE .....	3-16
3.2.8.3	ALTER DOMAIN .....	3-18
3.2.8.4	ALTER ENTITY .....	3-22
3.2.8.5	ALTER MAP .....	3-25
3.2.8.6	ALTER MODEL .....	3-30
3.2.8.7	ALTER RELATION .....	3-32
3.2.8.8	CHECK MODEL .....	3-35
3.2.8.9	COMBINE ENTITY .....	3-38
3.2.8.10	COMPARE MODEL .....	3-42
3.2.8.11	COPY ATTRIBUTE .....	3-44
3.2.8.12	COPY DESCRIPTION .....	3-48
3.2.8.13	COPY ENTITY .....	3-50
3.2.8.14	COPY MODEL .....	3-58
3.2.8.15	CREATE ALIAS .....	3-62
3.2.8.16	CREATE ATTRIBUTE .....	3-64
3.2.8.17	CREATE DOMAIN .....	3-66
3.2.8.18	CREATE ENTITY .....	3-68
3.2.8.19	CREATE MAP .....	3-71

TABLE OF CONTENTS (Continued)

3.2.8.20	CREATE MODEL .....	3-74
3.2.8.21	CREATE RELATION .....	3-76
3.2.8.22	CREATE VIEW .....	3-79
3.2.8.23	DEFINE DATABASE .....	3-82
3.2.8.24	DEFINE RECORD .....	3-85
3.2.8.25	DEFINE SET .....	3-88
3.2.8.26	DESCRIBE .....	3-91
3.2.8.27	DROP ALIAS .....	3-94
3.2.8.28	DROP ATTRIBUTE .....	3-96
3.2.8.29	DROP DATABASE .....	3-98
3.2.8.30	DROP DOMAIN .....	3-100
3.2.8.31	DROP ENTITY .....	3-103
3.2.8.32	DROP FIELD .....	3-106
3.2.8.33	DROP KEYWORD .....	3-109
3.2.8.34	DROP MAP .....	3-111
3.2.8.35	DROP MODEL .....	3-113
3.2.8.36	DROP RECORD .....	3-115
3.2.8.37	DROP RELATION .....	3-118
3.2.8.38	DROP SET .....	3-121
3.2.8.39	DROP VIEW .....	3-123
3.2.8.40	HALT .....	3-125
3.2.8.41	MERGE MODEL .....	3-127
3.2.8.42	RENAME .....	3-130
3.3	Performance Requirements .....	3-132
3.3.1	Programming Methods .....	3-132
3.3.2	Program Organization .....	3-132
3.3.3	Modification Consideration .....	3-132
3.3.4	Special Features .....	3-132
3.3.5	Expendability .....	3-132
3.4	Human Performance .....	3-133
3.5	Database Requirements .....	3-133
3.5.1	Database Overview .....	3-133
3.5.2	Relations Between Tables and Views .....	3-133
3.5.3	Detailed Description of Tables and Views .....	3-133
SECTION 4.0	QUALITY ASSURANCE PROVISIONS .....	4-1
4.1	Introduction and Definitions .....	4-1
4.2	Computer Programming Test and Evaluation .....	4-1
SECTION 5.0	PREPARATION FOR DELIVERY .....	5-1

LIST OF ILLUSTRATIONS

<u>Figure</u>	<u>Title</u>	<u>Page</u>
1-1	NDDL Functionality Matrix .....	1-3
3-1	NDDL Processor Interface .....	3-2
3-2	Organization of NDDL Functional Requirements .....	3-4

LIST OF TABLES  
CDM TABLES ACCESSED

<u>Table</u>	<u>Title</u>	<u>Page</u>
	ALTER ALIAS .....	3-15
	ALTER ATTRIBUTE .....	3-17
	ALTER DOMAIN .....	3-21
	ALTER ENTITY .....	3-24
	ALTER MAP .....	3-29
	ALTER MODEL .....	3-31
	ALTER RELATION .....	3-34
	CHECK MODEL .....	3-37
	COMBINE ENTITY .....	3-40
	COMPARE MODEL .....	3-43
	COPY ATTRIBUTE .....	3-47
	COPY DESCRIPTION .....	3-49
	COPY ENTITY .....	3-56
	COPY MODEL .....	3-60
	CREATE ALIAS .....	3-63
	CREATE ATTRIBUTE .....	3-65
	CREATE DOMAIN .....	3-67
	CREATE ENTITY .....	3-70
	CREATE MAP .....	3-73
	CREATE MODEL .....	3-75
	CREATE RELATION .....	3-78
	CREATE VIEW .....	3-81
	DEFINE DATABASE .....	3-84
	DEFINE RECORD .....	3-87
	DEFINE SET .....	3-90
	DESCRIBE .....	3-93
	DROP ALIAS .....	3-95
	DROP ATTRIBUTE .....	3-97

LIST OF TABLES (Continued)

<u>Table</u>	<u>Title</u>	<u>Page</u>
	DROP DATABASE .....	3-99
	DROP DOMAIN .....	3-102
	DROP ENTITY .....	3-105
	DROP FIELD .....	3-108
	DROP KEYWORD .....	3-110
	DROP MAP .....	3-112
	DROP MODEL .....	3-114
	DROP RECORD .....	3-117
	DROP RELATION .....	3-120
	DROP SET .....	3-122
	DROP VIEW .....	3-124
	HALT .....	3-129
	MERGE MODEL .....	3-130
	RENAME .....	3-132
ORACLE DATA DICTIONARY: COL TABLES .....		3-135

## SECTION 1

### SCOPE

#### 1.1 Identification

This specification establishes the development, test and qualification requirements of a computer program identified as the Neutral Data Definition Language Processor, known in this document as the NDDL Processor. The NDDL Processor is one configuration item of the Integrated Information Support System (IISS) Common Data Model (CDM) Subsystem.

#### 1.2 Functional Summary

The NDDL processor is a language used to manipulate and populate information in the Common Data Model (CDM) of the IISS System database. It provides the user with three modes of operation: (1) Batch Mode allows NDDL command files to be executed; (2) Interactive Mode allows the user to enter NDDL commands at a terminal; and (3) Forms Mode allows the user use of the IISS forms processor to display input and output screens of NDDL commands. The NDDL processor allows the user to populate and maintain the three schemas of the CDM: an external, conceptual and internal and the mappings between each. The NDDL also provides capabilities for manipulation of many IDEF-1 models and submodels needed during the process of developing the single integrated model of the conceptual schema. Only the integrated model may be mapped to the external and internal schemas. The NDDL was designed by a joint working group of IISS coalition members, described in the Integration Task Report, Reference 8. The language is modelled after SQL and the command features a combination of a few simple verbs (operators) along with the necessary parts of the CDM (objects). The functionality of NDDL is summarized in the matrix of Figure 1-1. The following notes refer to the foot notes of the matrix.

1. Internal Schema Objects are defined rather than created since IISS assumes internal schema describes actual, previously existing databases.
2. DESCRIBE serves the purpose of creating, altering and dropping descriptions for the applicable objects.
3. Aliases are maintained for entities and attributes only.
4. Keywords are maintained only for entities, attributes and relations. They can only be created when associated with entities, attributes or relations.
5. The noted objects can only be altered through use of DROP and CREATE operators.
6. ALTER commands generally have ADD, DROP and ALTER suboperators for subobjects.
7. Data items are created and dropped as subobjects of views.
8. Data types are created and dropped as subobjects of domains.
9. Data fields are created as subobjects of records.
10. Maps do not have names of their own and cannot be renamed or described.



O B J E C T

O P E R A T O R	A	A	D	D	D	D	D	E	F	K	M	M	R	R	S	V
	l i a s	t r i b u t e	a t a I t e m	a t a B a s e	a t a T y p e	e s c r i p t	o m i n i l y	n i l l w i r d	i l l w i r d	l e g a l l y	l e g a l l y	a p p l e d	l o c a l l y	l e g a l l y	l e g a l l y	l e g a l l y
Alter	x	x	5	5	5	1	x	x	5		x	x	5	x	5	5
Check												x				
Combine								x								
Compare												x				
Copy		x				x	x					x				
Create	x	x	7	1	8	2	x	x	9	4	x	x	1	x	1	x
Define				x									x		x	
Describe		x	x	x	x	n/a	x	x	x	x	10	x	x	x	x	x
Drop	x	x	7	x	8	2	x	x	x	x	x	x	x	x	x	x
Merge												x				
Rename		x					x	x		x	10	x		x		x
Halt																

Figure 1-1. NDDL Functionality Matrix

SECTION 2

DOCUMENTS

2.1 Reference Documents

1. General Electric Co., Test Bed System Requirement Document (Draft); SRD620140000, Revised 23 August 1982.
2. General Electric Co., Test Bed System Design Specification; SDS620140000, 7 February 1983.
3. ICAM Documentation Standards; IDS15012000A, 28 December 1981.
4. General Electric Co., IISS Software Development Guidelines/Conventions (Draft); 23 August 1982.
5. Structural Dynamics Research Corporation, IISS User Interface Management System Services User Manual; UM626144100A, July, 1983.
6. Structural Dynamics Research Corporation, IISS Form Processor Users Manual; UM620144200A, 5 December 1984.
7. Control Data Corporation, IISS Neutral Data Definition Language (NDDL) User's Guide (Preliminary Draft); 28 February 1985.
8. Hughes IISS Integration Task; 16 April 1984.
9. Softech, ICAM Architecture, Part II, Vol. V, Information Modelling (IDEF1); FTR110210000.
10. D. Appleton Co., CDM Administrator's Manual; UM620141000, March 1984.
11. D. Appleton Co., CDM1-IDEF, Model of the Common Data Model; CCS620141000, 15 May 1985.
12. General Electric Co., Quality Assurance Plan; QAP620144000, 4 January 1984.

13. D. Appleton Co., Embedded NDML Programmer's Reference Manual; PRM620141200, March, 1985.
14. Softech, Inc., NTM Programmer's Guide; UM620140001, July, 1984.

## 2.2 Terms and Abbreviations

Attribute Use Class: (AUC).

Application Interface: (AI) A collection of routines with the same calling sequences as the Forms Processor and Virtual Terminal callable routines that enables applications to be hosted on computers other than the host of the User interface.

Assertion: Predicate that applies to one or more attributes; checked after completion of an action to determine if the results should be committed.

Conceptual Schema: (CS).

Common Data Model Processor: (CDMP).

Common Data Model: (CDM) Describes common data application process formats, form definitions, etc. of the IISS and includes conceptual schema, external, internal schemas, and schema transformation operators.

Data Field: (DF) An element of data in the internal schema. Generally, it is by this name a DBMS will reference data.

Data Item: (DI) An element of data in the external schema. It is by this name that an NDML programmer references data.

Data Type: A specific computer representation of a domain.

Distributed Request Supervisor: (DRS) This IISS CDM Subsystem configuration Item controls the execution of distributed NDML queries and non distributed updates.

Domain: A logical definition of legal attribute class values.

Domain Constraint: Predicate that applies to a single domain.

External Schema: (ES).

Forms: Structured views which may be imposed on windows or other forms. A form is composed of fields where each field is a form, item, or window.

Forms Processor: (FP) A set of callable execution time routines available to an application program for form processing.

Internal Schema: (IS).

Integrated Information Support System: (IISS) A test computing environment used to investigate, demonstrate and test the concepts of information management and information integration in the context of Aerospace Manufacturing. The IISS addresses the problems of integration of data resident on heterogeneous databases supported by heterogeneous computers interconnected via a local Area Network.

Mapping: The correspondence of independent objects in two schemas: ES to CS or CS to IS.

NDDL User: The CDM administrator or his designated representative.

Network Transaction Manager: (NTM) Performs the coordination, communication and housekeeping functions required to integrate the Application Processes and System Services resident on the various hosts into a cohesive system.

Neutral Data Definition Languages: (NDDL) A language used to manipulate and populate information in the Common Data Model (CDM) or IISS System Database.

Neutral Data Manipulation Language: (NDML) A language developed by the IISS project to provide uniform access to common data, regardless of database manager or distribution criteria. It provides distributed retrieved and single node update.

ORACLE: Relational DBMS based on the SQL (Structured Query Language, a product of ORACLE Corp, Menlo Park, CA). The CDM is an ORACLE database.

Object: Named Common Data Model item; for example; entity class, relation class, attribute class.

Trigger: Action that is invoked at the commit completion of another action.

DS 620141100  
1 November 1985

User Interface: (UI) Controls the user's terminal and interfaces with the rest of the system.

Virtual Terminal Interface: (VTI) Performs the interfacing between different terminals and the UI. This is done by defining a specific set of terminal features and protocols which must be supported by UI software which constitutes the virtual terminal definition. Specific terminals are then mapped against the virtual terminal software by specific software modules written for each type of real terminal supported.

SECTION 3  
REQUIREMENTS

3.1 Computer Program Definition

The NDDL processor is the computer program that translates the command statements of this language and performs the operations requested, updating the CDM database. The NDDL language is non-procedural. The NDDL processor is essentially an interpreter, executing one command at a time, in the order presented by the user.

Each command is parsed for syntactic correctness. Control is transferred to the individual command processor for the semantic validation of the command. If all semantic checks are found to be correct, the database is updated or information retrieved.

3.1.1 System Capacities

The NDDL is designed to allow multiple users at the same time. Data limits are imposed only by the capacity of the DBMS. Processing speed limits are imposed by the speed of the computer and the number of other users and the speed and efficiency of the NTM subsystem and the IISS Forms Processor. A number of COBOL and C fixed size tables will be used to temporarily hold information. These limits can be changed very easily in a virtual memory environment.

3.1.2 Interface Requirements

The NDDL processor shall make use of the IISS Forms Processor for command input and shall allow batch input as well. The NDDL processor shall make use of IISS NDML wherever possible to retrieve data from the CDM native ORACLE wherever NDML is not sufficient.

3.1.2.1 Interface Block Diagram

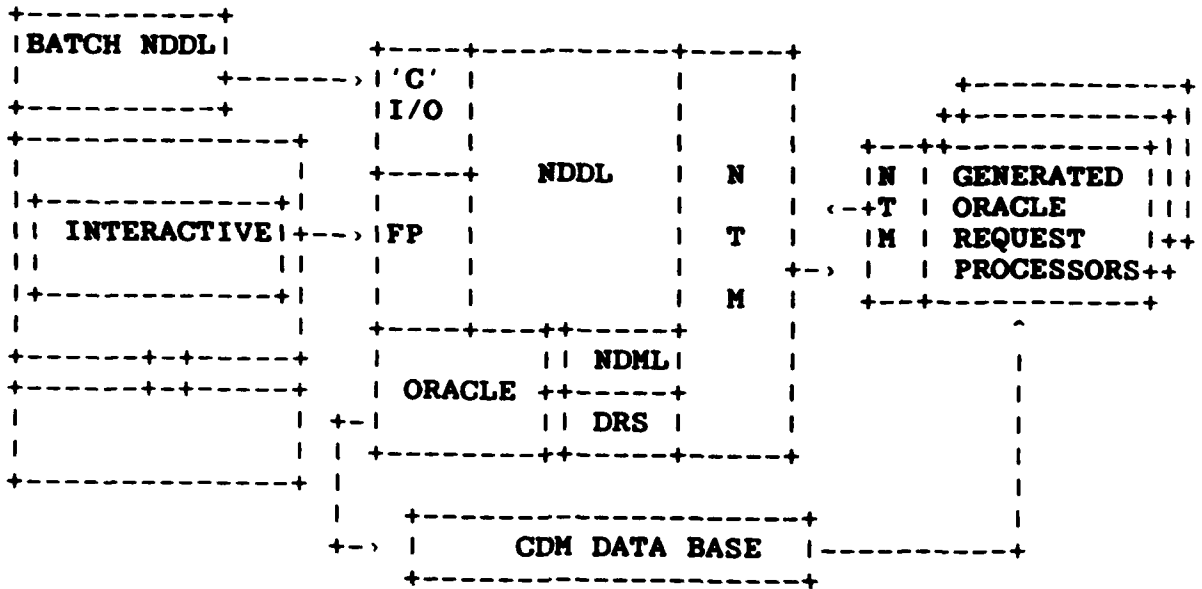


Figure 3-1. NDDL Processor Interfaces

### 3.1.2.2 Interface Requirements

The NDDL processor makes use of the IISS Forms Processor directly for forms interactive input. NDDL also makes use of the standard C input/output library to allow user non-forms interactive input or batch input via file redirection. Database access is through a combination of (1) ORACLE for update insert and delete and recursive searches not supported by NDML and (2) NDML for all other searches. The NDML routines are precompiled by the IISS NDML precompiler and ORACLE request processors are generated, which communicates with NDDL through the DRS which uses IISS NTM services.

It is a design goal to replace the use of ORACLE with NDML to achieve DBMS independence, and to allow the CDM database itself be distributed.

It is a design goal to make NDDL an application controlled by the IISS User Interface subsystem to make use of its capabilities. Currently there is an interface problem with the C library of I/O routines supplied with the IS/1 workbench and the UIMS.

### 3.2 Detailed Functional Requirements

This description of functional requirements is broken down into nine subfunctional areas. These areas are identified in the block diagrams, Figure 3-2 which includes the following paragraph numbers. The forty-two commands currently making up NDDL are each described in Section 3.2.8.



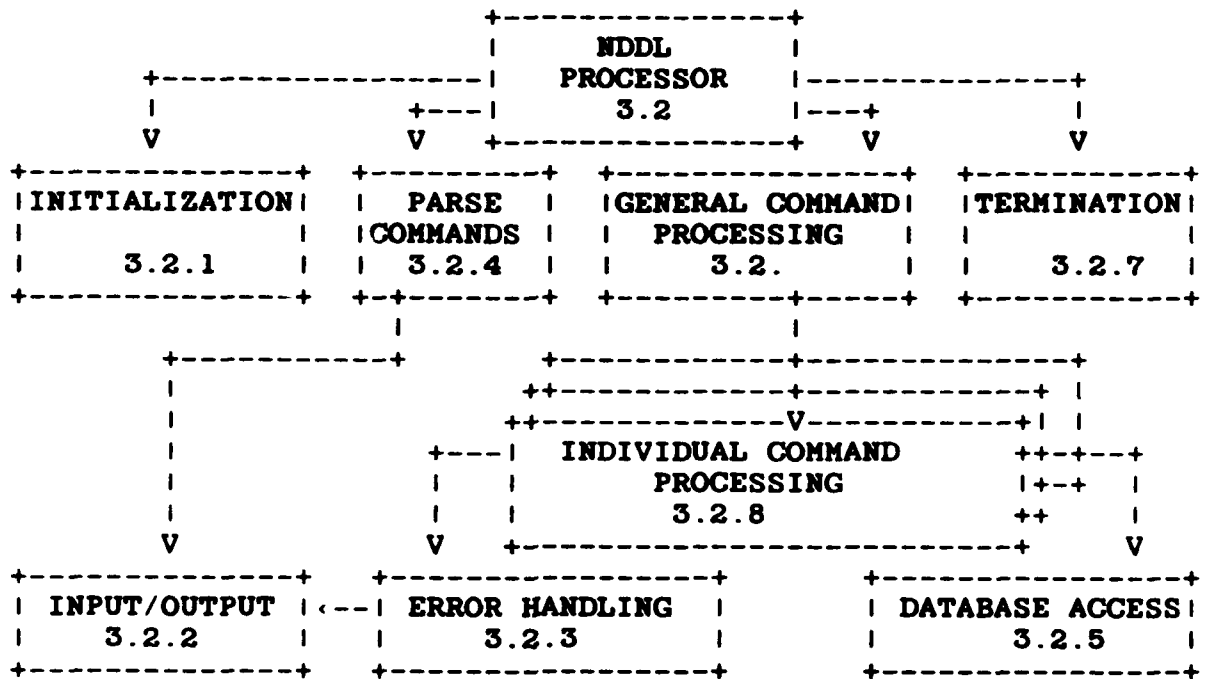


Figure 3-2. Organization of NDDL Functional Requirements

### 3.2.1 Initialization

#### A. Function:

Initialization will allow the NDDL processor to perform all initialization requirements with other subsystems and software environments.

#### B. CDM Requirements:

None

#### C. Processing:

1. Determine the processing mode: either interactive, through forms, or batched.
2. Initialize with NTM through use of INITEX service. In the future, this should be changed to INITAL when UI services are required.
3. If in forms mode, initialize to the forms processor by using the INITFP service, OPNFRM and ADDFRM to create the user's initial form.
4. Logon the ORACLE DBMS and open all cursors necessary. The logon data area and cursors will all be global data structures.
5. Initialize any other data structures necessary for any commands to their null, or initial state.
6. Initialize any other global variables such as current model, current database, etc.

### 3.2.2 Input/Output

#### A. Function:

Provide user input to the other components of the NDDL processor in an invisible manner, without respect to the means in which the input was obtained. Provide output for NDDL processor to the user through a standard interface to allow the same invisibility.

#### B. CDM Requirements:

None

C. Processing:

The standard C character input/output routines will be used. They will be modified, however, to recognize the current input/output mode. Batch mode will use the existing capability of the C library. Interactive mode will make use of forms processor calls. Because the forms processor can return many screens of information at a time, the modified input/output routines shall extract a single character at a time from the data structures. Since output consists of simple information messages to the user, PMSGLS forms processor calls can be used for all messages or PRINTF, if batch mode, to standard output. Output of generated NDDL uses the standard C file I/O primitives.

3.2.3 Error Handling

A. Function:

Provide a single standard means of communicating errors to the NDDL user. The interface shall be simple, readily usable and invisible to the particular input/output mode. The error handling shall also make database transaction rollback conditions simple to recognize. User requirements for command skipping after a semantic error shall be implemented.

B. CDM Requirements:

None

C. Processing:

Three major entry points to the error handling functionality of NDDL shall be established corresponding to the three types of errors. These are:

1. An entry point for "fatal" errors. These are errors from other subsystems or software than NDDL. These errors are to be handled in accordance with the IISS Error Handling Philosophy, Reference Number 12. A standard routine is called to log the message in a central place. These error conditions

shall also be communicated to the user as type 2 below.

2. An entry point for user errors. These are errors that are caused by the user and can be recovered by user action. An example may be creating an entity that already exists. This error handler must set a flag so at the end of the command, any database changes are backed out through a rollback procedure supplied by the DBMS or NDML.
3. An entry point for warning messages. These are indications of problems of user understanding, such as dropping a set that does not exist, or simple informative messages about actions that have occurred, such as "model altered".

It is the responsibility of the general command processor, Section 3.2.6, not to process commands in batch after a user error or fatal error has occurred. This prevents a later command from causing unpredictable harm.

#### 3.2.4 Parse Commands

##### A. Function:

The Parse Command subfunction of IISS will provide a mechanism for accepting user command input, validating correct syntax, reporting syntax errors and saving pertinent command information in data structures independent of the syntax.

##### B. CDM Requirements:

None

##### C. Processing:

This function will be provided through code generated by the UNIX tools YACC and LEX and interface routines provided as part of this function (UNIX is a trademark of Bell Labs).

1. LEX is a tool that generates lexical analyzers. Given a specification of the reserved words or tokens, LEX will generate a routine that will

accept user input and return control to the caller on each token recognized.

2. YACC is a tool that generates a parser that validates user input as matching the grammar or syntax of the language. The parser generated has the capability of calling user specified routines or code called "actions". YACC is commonly used in compiler construction. YACC uses a syntax specification of the NDDL commands and generates the NDDL parser. This specification is not treated as an IISS deliverable because to do so would require the user or target site to have the UNIX tools.
3. Token primitive routines will be developed that store user entered command data, or tokens, in a special data structure. This data structure is simply a matrix of pointers into a string of concatenated tokens. The columns of the matrix are called lists. Lists generally have like tokens; for example, all the keywords entered on a CREATE ENTITY command. The rows of the matrix are generally meaningless, unless the syntax defines a special correspondence between lists. An example is CREATE VIEW, where data items and attributes must match up.

The token primitives are the only kind of action statements used in the YACC input. It is conceivable that each entire command processor could be called as YACC action statements, but this is not the case. The design goal was to build command processors independent of the input mechanism, in this case, command syntax.

### 3.2.5 Database Access

#### A. Function:

This subfunction outlines the functional requirements of the database access used in the NDDL processor. All database access shall be for the ORACLE based CDM. All database access shall use the facilities of the IISS NDML wherever possible. The ORACLE SQL facilities may be substituted only if the NDML does not provide the needed functionality. Any use of ORACLE's SQL shall be

written in C. This is because the necessary logon database area and cursors must be kept in a global area to avoid the database access routines requiring any DBMS specific interface parameters. This is to allow eventual conversion of all ORACLE database access routines to NDML and achieve DBMS independence for the NDDL processor.

B. CDM Requirements:

The ORACLE DBMS must be used due to previous decisions on the initial DBMS to host the CDM.

C. Processing:

1. ORACLE SQL will be used for the following requirements:
  - 1.1 SELECT where sorting is required.
  - 1.2 SELECT where a Bill of Materials type recursive search is needed.
  - 1.3 INSERT operations. Insert modules will insert a single row at a time.
  - 1.4 DELETE operations.
  - 1.5 UPDATE (or MODIFY) operations.
2. COBOL embedded NDML will be used for all other database retrieval and verification modules. A distinction is made between verification or look up modules and modules expected to retrieve more than one row.
  - 2.1 The verification type module shall be called with the search parameters as inputs and the database value(s) found on the single row as output. Very often a zero valued object number will be used as a "no-find" status.
  - 2.2 For routines expected to find many rows, the routine will receive the simple search parameters as input as before. Within the NDML { and }, logic will be coded for processing each row. Very often calls to

other routines which may process a single row will be made. If row processing is simple enough, calls are not necessary.

These requirements promote DBMS independence and simplicity of data structures common to more than one module.

3. For purposes of database concurrency and integrity, the logical unit of work shall be defined to be a single NDDL command execution. That is, the command is wholly executed with the results as expected by the user or none of the command is executed. Therefore, an NDDL command can be considered a transaction.

### 3.2.6 General Command Processing

#### A. Function

General Command Processing will handle such functions as pre-command initialization, control of parsing, control of forms input/output, and post command control of database commit or rollback. It must also control parsing of commands that cannot be executed due to previous errors. This subfunction will also provide facilities for CDM object numbering and number reuse. The subfunction must provide for generalized access to the parser data structures.

#### B. CDM Requirements

Two tables necessary for object numbering will be used. These are: (1) NEXT\_NUMBER which contains the next number to be used for each object type; and (2) REUSABLE\_NUMBER which contains all the object numbers dropped and available for reuse.

#### C. Processing

1. The user input form must be displayed the first time in forms mode.
2. Each command entered by the user on the input screen must be processed; skipping commands after an error is encountered.

3. For a completed command, a count of errors must be displayed.
4. If these were errors, the entire screen must be redisplayed. Also, the previous set of error messages need to be blanked out and a "no errors" message displayed. If the user asked to refresh and keep his command on the screen, this too must be done.
5. The current database and model must also be kept on the screen.
6. If the user entered the quit key, a halt command must be generated and processed.
7. When a user has entered a command, the parser must be called. The return status of parsing must be examined.
8. If the command is to be processed, then a routine to effect the transfer of control to the proper command processor is executed.
9. After the individual command is executed, the current model and database must be established. If the command was successful, the database transactions are committed or, if unsuccessful, rolled back.
10. The CDM objects that shall be numbered follow. Each object type below has an object type number.

<u>OBJECT TYPE NUMBER</u>	<u>OBJECT TYPE</u>
1	MODEL
2	ENTITY
3	ATTRIBUTE
4	KEY CLASS
5	RELATION
6	TAG
7	DOMAIN
8	KEYWORD
9	VIEW
10	DATABASE
11	SET
12	DATA TYPE



13	DATA ITEM
14	DATA FIELD
15	RECORD

Objects are numbered to ease in renaming and to allow a central place for storing object descriptions.

Two subfunctions shall exist to promote consistent handling of these numbers.

10.1 Adding a reusable number shall make the number of a dropped object available for reuse by storing it in the CDM's REUSABLE\_NUMBER table.

10.2 Get next number will obtain a new, unused number for an object being created. It must first search the list of available numbers for this object type. If one is found, it of course must be deleted from the list of reusable numbers. If one is not found, the next available number is retrieved from the CDM's NEXT\_NUMBER table. This number is incremented and updated in the NEXT\_NUMBER table.

11. Finally, this subfunction must supply routines that allow the command processor to access the lists of user command tokens built by the parser. These functions shall allow access to the first token on the list, the next token from the list and accessing a token from one list corresponding to another list (same row). The functions should return a count of tokens on the list and an end of list indicator.

### 3.2.7 Termination

#### A. Function

Termination will allow the NDDL processor to perform all termination requirements with other subsystems and software environments.

B. CDM Requirements

None

C. Processing:

1. Close all ORACLE cursors and log off from ORACLE.
2. If the forms mode of input was used, use the FP service TERMFP.
3. Issue a call to send a finish up message to the DRS and to terminate NTM activities.

3.2.8 Individual Command Processing

The following subparagraphs outline the functional requirements of each command making up the NDDL. Consult the Table of Contents for a quick reference to a specific command.

3.2.8.1 ALTER ALIAS - Switch the primary and alias names of a conceptual attribute or entity.

A. Function:

Alter Alias performs the following functions:

1. changes the primary name of an attribute or entity to alias;
2. changes the alias name of an attribute or entity to primary.

B. CDM Requirements:

1. The primary name of the attribute or entity must exist in the current model.
2. The alias name of the attribute or entity must exist in the current model.

C. Processing:

1. Alter Alias verifies that the primary and alias names to be switched exist in the current model.

DS 620141100  
1 November 1985

2. If attribute names are being switched, the primary and alias entries in the ATTRIBUTE\_NAME table are updated.
3. If entity names are being switched, the primary and alias entries in the ENTITY\_NAME table are updated.

DS 620141100  
1 November 1985

CDM TABLES ACCESSED - ALTER ALIAS (S=SQL N=NDML)

<u>TABLE NAME</u>	<u>SELECT</u>	<u>MODIFY</u>	<u>INSERT</u>	<u>DELETE</u>
ENTITY_CLASS	N(VERNME)			
ENTITY_NAME	N(VERNME)	S(UPDECAL)		
ATTRIBUTE_CLASS	N(VERNMA)			
ATTRIBUTE_NAME	N(VERNMA)	S(UPDACAL)		

**3.2.8.2 ALTER ATTRIBUTE - Alter a Conceptual Attribute**

**A. Function:**

Alter Attribute performs the following functions:

1. change a domain name for an attribute;
2. add keywords to an attribute;
3. drop keywords from an attribute.

**B. CDM Requirements:**

1. The attribute to be altered must exist in the current model.
2. If the domain is to be modified, the new domain must exist.
3. If a keyword is to be dropped, it must exist.

**C. Processing:**

1. Alter Attribute verifies that the attribute to be altered exists.
2. If the domain is to be changed, the existence of the new domain is verified and the ATTRIBUTE\_CLASS table is modified to contain the new domain number.
3. If a keyword is to be dropped, a check is performed to verify that the keyword is assigned to the attribute. If so, the keyword is deleted from the AC\_KEYWORD table.
4. If a keyword is to be added to an attribute, the keyword table is searched to determine whether the keyword exists. If not, the new keyword is inserted into the keyword table. The new keyword is then inserted into the AC\_KEYWORD table, if it did not already exist there.

DS 620141100  
1 November 1985

CDM TABLES ACCESSED - ALTER ATTRIBUTE (S-SQL,N-NDML)

<u>TABLE NAME</u>	<u>SELECT</u>	<u>MODIFY</u>	<u>INSERT</u>	<u>DELETE</u>
AC_KEYWORD (DELKWAC)	N (ADDWA)		S INSKWAC)	S
ATTRIBUTE_CLASS	N (VERATT)	S (UPDAC)		
ATTRIBUTE_NAME	N (VERATT)			
DOMAIN_CLASS	N (VERDOM)			
KEYWORD	N (VERKW)		S (INSKW)	

3.2.8.3 ALTER DOMAIN - Alter the definition of a domain in the CDM.

A. Function:

Alter Domain allows the NDDL user to perform the following modifications to the definitions of existing domains:

1. addition of non-standard data types;
2. deletion of non-standard data types;
3. changing the meta data description of existing data types of the domain (i.e. changing the type, size and number of decimal digits);
4. promoting a non-standard data type to standard, converting the former standard to non-standard.

B. CDM Requirements:

The domain name referenced must be found in the CDM. Any data types to be dropped or altered must already be defined for the domain. Any data types to be added must not already be defined anywhere else in the CDM. There must always be a standard data type for the domain, i.e. the current standard data type cannot be dropped.

C. Processing:

1. The user entered domain name to be altered is verified to be in the CDM. With this number, each of the data type changes can be processed.
2. For each user data type request, determine if its an ADD, DROP or ALTER.
  - 2.1 For an ADD or ALTER, the legal data types are checked. They must be SIGNED, UNSIGNED, INTEGER, FLOAT, PACKED, or CHARACTER. The user does not enter this for a DROP.
  - 2.2 For an ADD and optionally for an ALTER, the size and number of decimal digits are checked. They must both be numeric and the decimal

digits must not exceed the size. They are not specified for a DROP.

2.3 If the user has requested to alter a data type:

2.3.1 The data type is verified to exist for the domain and be either standard or non-standard. The user is warned if it cannot be found. If it is a standard, only type, size and number of decimals may be changed.

2.3.2 For standard data type, or a change to only the type, size and number of decimals, this change is recorded in the USER\_DEF\_DATA\_TYPE table.

2.3.3 Otherwise, the user has requested a switch from non-standard to standard. In this case, the old standard data type name is fetched and is changed to non-standard and the name specified by the user is changed to become the standard data type.

2.4 If the user has requested to drop a data type:

2.4.1 The data type name is verified to be in the domain.

2.4.2 If the data type is found to be standard, the user is informed that it cannot be dropped.

2.4.3 If the data type is non-standard, then any usage of this data type is checked.

2.4.3.1 The database is searched to find any references by a data field.

2.4.3.2 The database is searched to find any references by a data item.

2.4.3.3 The database is searched to find any references by an attribute class. This is probably unnecessary since it has been checked to be non-standard in 2.4.2 and



only standards can map to attributes.

- 2.4.4 If it was not referenced elsewhere, data type and its description text is deleted.
- 2.5 If the user has requested to add a data type:
  - 2.5.1 For a standard data type, the database is searched for an existing standard data type. If one is not found:
    - 2.5.1.1 The data type name is checked to see that it was not used for some other domain. If not,
    - 2.5.1.2 The data type is checked to be valid by using a database look up in the table DATA\_TYPE. If ok,
    - 2.5.1.3 The data type information is stored in the CDM table USER\_DEF\_DATA\_TYPE, with a unique object number, as standard for this domain.
  - 2.5.2 For a non-standard data type, the data type name is checked as in step 2.5.1.1.
    - 2.5.2.1 The data type is checked as in 2.5.1.2.
    - 2.5.2.2 The data type information is stored in the CDM table USER\_DEF\_DATA\_TYPE with a unique object number as non-standard (or "USER") for this domain.

DS 620141100  
1 November 1985

CDM TABLES ACCESSED - ALTER DOMAIN

(S=SQL N=NDML)

<u>TABLE NAME</u>	<u>SELECT</u>	<u>MODIFY</u>	<u>INSERT</u>	<u>DELETE</u>
ATTRIBUTE_CLASS	N(VERACDT)			
DATA_FIELD	N(VERDFDT)			
DATA_ITEM	N(VERDIDT)			
DATA_TYPE	N(VERTYP)			
DOMAIN_CLASS	N(VERDOM)			
USER_DEF_DATA_TYPE		S(UPDTDT)	S(INS DT)	S(DELDT)
USER_DEF_DATA_TYPE	N(VERDT)	S(UPDIND)		
USER_DEF_DATA_TYPE	N(VERDTD)			
USER_DEF_DATA_TYPE	N(VERS DT)			

3.2.8.4 ALTER ENTITY - Alter a conceptual entity

A. Function:

Alter Entity performs the following functions:

1. add/drop key classes for the entity being altered;
2. add/drop owned attributes for the entity being altered;
3. add/drop associated keywords for the entity class being altered.

B. CDM Requirements:

1. The entity to be altered must exist in the current model.
2. If owned attributes are to be added, the attribute must exist in the current model. If owned attributes are to be dropped, they must be owned by the entity being altered.
3. If a key class is to be dropped, it must be a key class for the entity being altered.
4. If a keyword is to be dropped, it must be associated with the entity being altered.

C. Processing:

1. The Alter Entity process verifies that the entity to be dropped exists in the current model. If it does not exist, an error is issued and processing is terminated.
2. If key classes are being added, a new occurrence of key class is added to the entity. A new occurrence of attribute use class is created for each attribute named as part of the key, if one does not exist for the entity. A new occurrence of key class members is created for the entity for each attribute named in the key class clause.
3. If owned attributes are being added for the entity,

the existence of the attribute class is determined within the current model. If the attribute class does not exist, an error is issued and processing is terminated. If they do exist, each attribute class is created as an owned attribute class and attribute use class for the entity.

4. If a keyword is to be added to the entity class, the keyword table is searched to determine whether the keyword exists. If it does not exist, the new keyword is inserted into the keyword table. The new keyword is then associated with the entity class.
5. If key classes are being dropped, the existence of the key class for the entity being altered is determined. If it does exist, the key class and attributes inherited via the migrated keys and key class members are dropped. If the key class being dropped is from a complete relation, the complete relation is also deleted.
6. If owned attributes are being dropped, they are verified to determine if they belong to the entity being altered. If they belong to the entity, the owned attribute class occurrence and the attribute use class for each attribute named is deleted from the entity being altered.
7. If keywords are to be dropped, a check is performed to verify that the keyword is associated with the entity being altered. If so, the keyword association is deleted from the entity.

DS 620141100  
1 November 1985

CDM TABLES ACCESSED - ALTER ENTITY

(S=SQL,N=NDML)

<u>TABLE NAME</u>	<u>SELECT</u>	<u>INSERT</u>	<u>MODIFY</u>	<u>DELETE</u>
ATTRIBUTE_CLASS	N(VERATT)			
ATTRIBUTE_NAME	N(VERATT)			
ATTRIBUTE_USE_CL	N(DELOAC) N(VERAUC)	S(INSAUC)		S(DELAUCL)
COMPLETE_RELATION	N(VERCRC)			S(DELCMPR)
EC_KEYWORD	N(ADDKWE)			S(DELKWEC)
ENTITY_CLASS	N(VERENT)			
ENTITY_NAME	N(VERENT)			
INHERITED_ATT_USE	S(DELMIGK)			S(DELIAUC)
KEYWORD	N(VERKW)	S(INSKW)		
KEY_CLASS	N(VERKC)	S(INSKC)		S(DELKC)
KEY_CLASS_MEMBER	N(DRPMGKM)	S(INSKCM)		S(DELKCM)
OWNED_ATTRIBUTE	N(DRPAC) N(VEROAC)	S(INSOAC)		

**3.2.8.5 ALTER MAP - Modify a CS-IS Mapping**

**A. Function:**

Alter Map allows the user to perform the following functions:

1. add a data field mapping to an attribute use class (AUC) to data field map;
2. add a set mapping to an AUC to set map;
3. drop a data field mapping from an AUC to data field map;
4. drop a set mapping from an AUC to set map;
5. change a secondary AUC to data field mapping to primary and the previous primary mapping to secondary;
6. change the data type name for an AUC to data field mapping;
7. change the value in an AUC to set mapping;
8. add a set mapping to a relation class to set map;
9. drop a set mapping from a relation class to set map.

**B. CDM Requirements:**

The map to be altered must exist in the CDM. In addition, all database names, record names, data field names, data type names, and set names referenced must exist in the CDM.

**C. Processing:**

Different rules apply depending on the Alter Map option chosen.

1. For an AUC to data field map, the following rules apply:

1.1 ALTER ADD

- 1.1.1 The AUC must not have been previously mapped to a set.
- 1.1.2 The AUC must not have been previously mapped to a data field.
- 1.1.3 If no data type name is entered, the standard data type for the AUC's domain is used.
- 1.1.4 Only one primary mapping may exist for an AUC.
- 1.1.5 Multiple secondary mappings may exist if there is a pre-existing primary mapping.
- 1.1.6 If a primary or secondary mapping is not specified, the default is secondary.

If all rules are obeyed, a PROJECT\_DATA\_FIELD entity is inserted into the CDM.

## 1.2 ALTER DROP

- 1.2.1 If secondary AUC to data field mappings exist, a primary AUC to data field map cannot be dropped.

If the above rule is obeyed, a PROJECT\_DATA\_FIELD entity is deleted from the CDM.

## 1.3 ALTER ALTER

No special validations are performed for the ALTER ALTER option for AUC to data field mappings. This option allows a secondary mapping to be changed to primary and the previous primary mapping to secondary. This option also allows the data type name to be changed. In both cases, PROJECT\_DATA\_FIELD entities are modified.

- 2.0 For an AUC to set map, the following rules apply:

## 2.1 ALTER ADD

- 2.1.1 A data field mapping must not exist for the AUC.
- 2.1.2 The set to be mapped to must have a single record type for its members.
- 2.1.3 The set to be mapped to must not have been previously mapped from a relation class or another AUC.
- 2.1.4 All AUC to set maps must map to the same database for a particular AUC.
- 2.1.5 All AUC to set maps must contain a value which must be unique for a particular AUC.
- 2.1.6 All sets mapped to from an AUC must have the same record type as its owner.

If the above rules are obeyed, an AUC\_ST\_MAPPING entity is created.

## 2.2 ALTER DROP

No special validations are performed for the ALTER DROP option for an AUC to set mapping. An AUC\_ST\_MAPPING entity is deleted.

## 2.3 ALTER ALTER

The AUC value must be unique for all mappings from a particular AUC.

If the above rule is obeyed, an AUC\_ST\_MAPPING entity is modified.

## 3.0 For a relation class to set mapping, the following rules apply:

### 3.1

- 3.1.1 The set must not have been previously mapped.



3.1.2 The member record name must be specified if the set being mapped to is a multi-member set.

If the above rules are obeyed, an RC\_BASED\_REC\_SET entity is created.

### 3.2 ALTER DROP

No special validations are performed for the ALTER DROP option for a relation class to set mapping. An RC\_BASED\_REC\_SET entity is deleted.

### 3.3 ALTER ALTER

The ALTER ALTER option is invalid for relation class to set maps.

CDM Tables Accessed - Alter Map

<u>TABLE NAME</u>	<u>SELECT</u>	<u>MODIFY</u>	<u>INSERT</u>	<u>DELETE</u>
ATTRIBUTE_CLASS	N(FINDDOM)			
ATTRIBUTE_USE_CL	N(VERAUC)			
AUC_ST_MAPPING	N(FNDASA) N(VOMAPS) N(VERASH) N(FNDASH) N(CHLTAUCV)	S(ALTSMAP)	S(INSAUCS)	S(DELIASM)
DATA_BASE	N(VERDB) N(VERDF)			
DATA_FIELD	N(VERDF)			
ENTITY_CLASS	N(VERENT)			
ENTITY_NAME	N(VERENT)			
PROJECT_DATA_FIELD	N(PDFSRCH) N(VERPDF) N(GETMAPC)	S(ALTSMAP)	S(INSPDF)	S(DELIPDF)
RC_BASED_REC_SET	N(VERRCBS) N(VERRCMP) N(FNDRCM)		S(INSRCRS)	S(DELIRCS)
RECORD_SET	N(VERSMS) N(VOMAPS)			
RECORD_TYPE	N(VERDF)			
RELATION_CLASS	N(VERRC)			
SET_TYPE_MEMBER	N(FND1MEM)			
USER_DEF_DATA_TYPE	N(VERSDT) N(VERDTD)			

3.2.8.6 ALTER MODEL - Creates a current model for a NDDL session using modeling commands.

A. Function:

Alter Model performs the following functions:

1. updates the date of the model showing the model change date;
2. creates a current model for a NDDL session;
3. changes the status of the model to "UNCHECKED".

B. CDM Requirements:

Alter Model requires that the model to altered exists in the CDM.

C. Processing:

1. The Alter Model process verifies that the model was created previously. If the model does not exist, an error occurs and an error message is issued.
2. The model being altered becomes the current model for a NDDL modeling session with all subsequent model processing identified with the model. The CDM MODEL\_CLASS table is updated to show the system date as the date the model was updated and changes the model status to "UNCHECKED".

DS 620141100  
1 November 1985

CDM TABLES ACCESSED - ALTER MODEL

(S-SQL,N-NDML)

<u>NAME</u>	<u>SELECT</u>	<u>MODIFY</u>	<u>INSERT</u>	<u>DELETE</u>
MODEL_CLASS	VERMOD(NDML)			
MODEL_CLASS		UPDMOD		

3.2.8.7 ALTER RELATION - Alter a conceptual relation class.

A. Function:

Alter Relation performs any or all of the following functions:

1. change to cardinality of an existing relation class;
2. migrate the key class of the independent entity to the dependent entity, creating a complete relation and inherited attribute use classes;
3. assign new tag names to the key class members migrated to the dependent entity;
4. associate one or more keywords with the relation class;
5. drop the key class from the relation and from the dependent entity and all subsequent entities that inherited the key class;
6. delete any empty key classes that results from dropping a key class migration.

B. CDM Requirements:

1. Key class for independent entity must exist.
2. Key class members for independent entity must exist
3. An attribute use class for each key class member must exist.
4. Relation class must exist.
5. Independent entity class must exist.
6. Dependent entity class must exist.
7. If a key class is to be migrated to the dependent entity, the key class must not have been previously migrated to the dependent entity.
8. If a key class is to be dropped from the dependent

entity and all subsequent entities, the key class must have been previously migrated.

C. Processing:

Processing varies depending on the options chosen by the user. If an error is detected, processing continues with the next option on the command.

1. **CARDINALITY**

Any cardinalities specified replace the original values established when the relation was created, unless an error is detected, a warning message is generated and the cardinality defaults to its original value.

2. **ADD MIGRATES**

An attribute use class and an inherited attribute use class for the dependent entity is created for each key class member migrated to the dependent entity class. If the set phrase is specified, TAG\_NAME1 (the independent entity is tag name) is migrated to the dependent entity with the new name TAG\_NAME2. A complete relation class occurrence is created. If a keyword is specified, the keyword is created in the CDM if it doesn't already exist and a relation class keyword occurrence is created.

3. **DROP MIGRATES**

The complete relation occurrence for the relation class is deleted. The attribute use class, and inherited attribute use class originally created for each key class member migrated to the dependent entity class are deleted. In addition, the attribute use classes and inherited attribute use classes created for each key class member migrated to lower level dependent entity classes are also deleted. Then all key classes and complete relations which become empty due to the deletion of migrated key classes members, are deleted. Finally, any text descriptions for empty key classes are deleted. If keywords are specified, the relation class keyword occurrence are also deleted.

CDM TABLES ACCESSED - ALTER RELATION

(S=SQL N=NDML)

<u>TABLE NAME</u>	<u>SELECT</u>	<u>MODIFY</u>	<u>INSERT</u>	<u>DELETE</u>
ATTRIBUTE_USE_CL	N(ADDMIG) N(VERAUC)		S(INSAUC)	S(DELAUCL)
COMPLETE_RELATION	N(VERRC)		S(INSCRC)	S(DELCMPR)
DESC_TEXT				S(DELTEXT)
ENTITY_CLASS	S(DELMTKC) N(VERENT)			
INHERITED_ATT_USE	S(DLMIGRC)		S(INSTAUC)	S(DELIAUC)
KEY_CLASS	N(VERKC) S(DELMTKC)			S(DELKC)
KEY_CLASS_MEMBER	N(ADDMIG) S(DRPMGRC) S(DELMTKC)			S(DELKCMT)
KEYWORD	N(VERKW)		S(INSKW)	
RC_KEYWORD	N(ADDKWR)		S(INSKWRC)	S(DELKWRC)
RELATION_CLASS	N(VERRC)		S(UPDTRC)	

3.2.8.8 CHECK MODEL - Determines if the model conforms to specified IDEF1 rule

A. Function:

The check model performs the following functions:

1. verifies that the model exists in the CDM;
2. verifies that the model has one or more entities;
3. verifies that the entities have at least one attribute;
4. verifies that the model follows the specified IDEF1 rules (see rules under Processing);
5. updates the model in the CDM to show that it is a "checked" model and the date it was checked.

B. Requirements:

The check model process requires that the model exist in the CDM. (See Rules under Processing).

C. Processing:

The check model process determines if the model follows the following IDEF-1 rules.

Rules:

1. nonon-specific relations are allowed (independent cardinality greater than one).
2. no incomplete relation classes, (key has not been migrated).
3. each entity must have at least one attribute use class.
4. each owned attribute class must have a domain and that domain must have a standard data type.
5. a key class must be defined for each entity class.



6. multiple key classes of an entity class must not be subsets of one another.
7. no one to one relations.
8. no dependency loops e.g. A→B→C→D→B.
9. at least one entity must exist in the model.

The following rules cannot be checked for the model:

1. one to none or one relationships imply identical keys.
2. key uniqueness throughout the model is not checked, i.e. no two entity classes may have the same key class unless they are related to each other with a one or none or one relation.

The processing verifies the existence of the model. The process then selects each entity class belonging to the model and checks the relation classes, key classes, and attributes.

Next the process checks the hierarchical dependencies both up and down to determine if there are any dependency loops within the model.

If all rules have been followed the CDM MODEL\_CLASS table is updated to reflect the date and the model status of CHECKED.

<u>CDM TABLES ACCESSED - CHECK MODEL</u>		<u>(S-SQL, N-NDML)</u>		
<u>TABLE NAME</u>	<u>SELECT</u>	<u>MODIFY</u>	<u>INSERT</u>	<u>UPDATE</u>
ATTRIBUTE_CLASS	N(CHKATT)			
ATTRIBUTE_USE_CL	N(CHKATT)			
ENTITY_CLASS	S(CHKLOOP) N(GETECS)			
ENTITY_NAME	N(RETRECP) N(GETECS)			
KEY_CLASS	S(CHKKEYS)			
KEY_CLASS_MEMBER	S(CHKKEYS)			
MODEL_CLASS	S(CHKLOOP) N(VERMOD)			UPDMOD
OWNED_ATTRIBUTE	S(CHKATT)			
RELATION_CLASS	N(CHKREL) S(CHKLOOP) S(TLOOPCK) S(BLOOPCK)			

**3.2.8.9 COMBINE ENTITY - Combine two conceptual entities.**

**A. Function:**

**Combine Entity performs the following functions:**

1. combine two entities that exist either within the same model (intra-model) or between two models (inter-model);
2. generate NDDL commands on a file to populate the to-model entity with the attributes, relations, aliases, keywords, keys, key class members associated with the from-model entity.

**B. CDM Requirements:**

1. If the to-model is not specified, an inter-model combine is assumed, and the to-model must exist in the CDM.
2. If the to-model and from-model are specified, both models must exist in the CDM.
3. The two entities to be combined must exist in the model(s).

**C. Processing:**

1. If it is an intra-model combine, to-model defaults to the from-model.
2. First, verify that the two entities to be combined exist in the from-model and to-model. Processing halts if any of the verification checks fail. The NDDL commands to combine the from-entity and to-entity are generated in a user defined file. This file is created if it did not previously exist, or opened and appended to if it did already exist.
3. Now determine if a relation exists between the to-entity and from-entity. If one does, generate a command to drop this relation. Also, if it is an intra model combine generate a command to delete the from-entity.

4. The from-entities keys and key class members are saved in a temporary key list, in order to migrate the keys via new relations that will be created after the from-entity has been combined into the to-entity.
5. Generate an "Alter entity and owned attributes..." for all the attributes that belonged to the from-entity. If the user specified that he wanted keywords, aliases and/or descriptions, NDDL commands are generated for the same. The from-entity name is generated as an alias for the to-entity.
6. Next, select all relations in the from-model where the from-entity is the dependent entity in the relation. If this from-independent entity(s) exists in the to-model, generate Create relation ... migrates...commands for the same. The key class that was inherited via this relation has to be generated for the to-entity. Generate an Alter Entity add key...for the inherited key class.
7. Next, select all the relations in the from-model where the from-entity is the independent entity in the relation. If this from-dependent entity exists in the to-model, generate a create relation...migrates...command using the information stored earlier in the temporary key list.
8. Commands are also generated to associate keywords and descriptions with the relation if any exist. Finally, close the user defined file at the end of processing.

CDM TABLES ACCESSED - COMBINE ENTITY (S=SQL, N=NDML)

<u>TABLE NAME</u>	<u>SELECT</u>	<u>MODIFY</u>	<u>INSERT</u>	<u>DELETE</u>
AC_KEYWORD	N(GENAKW)			
ATTRIBUTE_CLASS	N(CMBOA) N(VERATT)			
ATTRIBUTE_NAME	N(CMBOA) N(VERATT) N(CMBACAL)			
ATTRIBUTE_USE_CL	N(BLKCL1) S(SELIAUC)			
COMPLETE_RELATION	N(VERRCC)			
DESC_TEXT	N(GENDESC)			
DOMAIN_CLASS	N(CMBOA)			
EC_KEYWORD	N(CMBEKW) N(VERKWE)			
ENTITY_CLASS	N(CMBALI) N(VERENT) N(BLKCL1) S(SELIAUC)			
ENTITY_NAME	N(VERENT) N(SELECNM) N(CMBALI) N(VERALI)			
INHERITED_ATT_USE	S(SELIAUC)			
KEY_CLASS	N(BLKCL1)			
KEY_CLASS_MEMBER	N(BLKCL1)			
KEYWORD	N(CMBRKW) N(CMBEKW) N(VERKWE) N(VERKWR) N(GENAKW)			

DS 620141100  
1 November 1985

CDM TABLES ACCESSED - COMBINE ENTITY (S-SQL, N-NDML)

<u>TABLE NAME</u>	<u>SELECT</u>	<u>MODIFY</u>	<u>INSERT</u>	<u>DELETE</u>
MODEL_CLASS	N(VERMOD)			
OWNED_ATTRIBUTE	N(CMBOA)			
RC_KEYWORD	N(CMRKW) N(VERKWR)			
RELATION_CLASS	N(DEPFROM) N(INDFROM) N(VERRC) N(SELRCNM)			

3.2.8.10 COMPARE MODEL - Compare two IDEF1 models.

A. Function:

Compare model to see if their entity name, attribute name, entity keywords, attribute keywords and relation keywords match each other.

B. CDM Requirements:

The two models to be compared must exist.

C. Processing:

1. First, verify the existence of both models, if either of these two models does not exist, flag a user error; otherwise, do the following:
  - 1.1 Compare entity classes based on identical names.
  - 1.2 Compare attribute classes based on identical names.
  - 1.3 Compare entity keywords to determine if entities from both models use the same keyword.
  - 1.4 Compare attribute keywords in the same manner as entities.
  - 1.5 Compare relation class keywords.
2. For each successful comparison above, a message will be printed out to indicate a match is found in two models.

DS 620141100  
1 November 1985

CDM TABLES ACCESSED - COMPARE MODEL

(S=SQL, N=NDML)

<u>TABLE NAME</u>	<u>SELECT</u>	<u>MODIFY</u>	<u>INSERT</u>	<u>DELETE</u>
AC_KEYWORD	N(RETACKW)			
ATTRIBUTE_CLASS	N(RETRAC1)			
ATTRIBUTE_NAME	N(RETRAC1)			
ENTITY_CLASS	N(RETREC1) N(VERENT) N(RETECKW) N(RECKW2) N(RELKW)			
ENTITY_NAME	N(RETREC1) N(VERENT) N(RETRECP)			
KEYWORD	N(RETACKW) N(RETECKW) N(RETACKW)			
MODEL_CLASS	N(VERMOD)			
RC_KEYWORD	N(RETACKW) N(RRCKW2)			
RELATION_CLASS	N(RETACKW) N(RRCKW2) N(GETRCID)			



3.2.8.11 COPY ATTRIBUTE - Copies an attribute and all associated information from one model to another model (inter-model) or within a model (intra-model).

A. Functions:

Copy Attribute performs the following functions:

1. verifies that the model specified exists;
2. copies an attribute within a model or to another model;
3. verifies that the new attribute does not exist in the current model;
4. when indicated, copies all description text, aliases, and keywords related to the attribute;
5. optionally, places the created NDDL commands in a file for later use;
6. optionally, interactively performs the intra-model copy.

B. CDM Requirements:

The Copy Model process requires that the from-model and the attribute exist in the CDM database.

C. Processing:

The following rules apply to the copy attribute process:

INTER-MODEL Copy

1. The from-model must be specified and it must exist in the CDM database.
2. The command must include the file clause that will contain the generated NDDL commands.

INTRA-MODEL Copy

1. The second or new attribute clause must be specified and must not exist in the current model.

2. The file clause maybe used.

#### General Rules

1. The except clause, when used, indicates what items associated with the attribute are not be copied. If the except clause is omitted all keyword, aliases, and textual descriptions of the attribute are copied.
2. The process verifies that the current model exists in the CDM database. Additionally, the attribute to be copied is verified and its domain returned. If either attribute or model is not found, an error message is issued and the processing terminates.
3. The processing determines whether the copy is to be interactive or a copy to a file. If the process is interactive only, the intra-model copy may be processed.

#### Interactive Process:

1. The interactive process verifies that the new or second attribute does not exist in the current model. If the attribute does not exist, the process inserts the new or second attribute into the ATTRIBUTE\_CLASS and ATTRIBUTE\_NAME CDM tables.
2. The following processing depends upon the use of the except clause. Any entry in the except will exclude that entry from processing. The copied attribute aliases will be retrieved and copied to the new attribute in the ATTRIBUTE\_NAME CDM table.
3. The copied attributes keywords will be retrieved and copied to the new attribute in the AC\_KEYWORD tables.
4. The copied attributes textual descriptions will be retrieved and copied to the new attribute in the DESC\_TEXT table.

#### COPY TO FILE PROCESS

DS 620141100  
1 November 1985

1. The copy to a file process will write generated NDDL commands to the file specified for either an inter- or an intra-model attribute copy.
2. The process verifies for an intra-model copy, that the new attribute does not exist in the current model. NDDL commands to create the attribute are generated and written to the user specified file. If the except keyword was not specified, the NDDL create attribute keyword option is included if a keyword exist for the copied attribute. If the except description was not specified, describe commands are generated and written to the file. If the except alias was not specified, create alias commands are generated and written to the file.

DS 620141100  
1 November 1985

CDM TABLES ACCESSED - COPY ATTRIBUTE

(S=SQL, N=NDML)

<u>TABLE NAME</u>	<u>SELECT</u>	<u>INSERT</u>	<u>MODIFY</u>	<u>DELETE</u>
AC_KEYWORD	N(WRTACKW) N(GENAKW)		S(INSKWAC)	
ATTRIBUTE_CLASS	N(VERACNM) N(VERATT)			
ATTRIBUTE_NAME	N(VERATT) N(WRTALI) N(VERACNM) N(FCOPATT)		S(INSACNM)	
DESC_TEXT	N(GENDESC)			
	S(WRTDESC)		S(WRTDESC)	
DOMAIN_CLASS	N(VERACNM)			
KEYWORD	N(GENAKW)			
MODEL_CLASS	N(VERMOD)			

3.2.8.12 COPY DESCRIPTION - Copy CDM object descriptions.

A. Function:

This is a partial description copy. Only the description lines of the identified description type of the given object will be copied, rather than all description types.

B. CDM Requirements:

The two objects identified must exist, and the description type and the description of the object must exist. If the DESC\_TEXT of the second object does not exist, the DESC\_TEXT of the first object can be copied.

C. Processing:

1. Verify the existence of the model. If it does not exist, flag a user error.
2. Verify the existence of description type. If it does not exist, flag a user error.
3. Verify the existence of both models. If either of these two models does not exist, flag a user error.
4. Verify the existence of description text of the first object. If it does not exist, flag a user error.
5. Verify the non-existence of description text of the second object. If it does exist, flag a user error.
6. Finally, copy the description text of the first object to the second object.

DS 620141100  
1 November 1985

CDM TABLES ACCESSED - COPY DESCRIPTION

S(S=SQL, N=NDML)

<u>TABLE NAME</u>	<u>SELECT</u>	<u>MODIFY</u>	<u>INSERT</u>	<u>DELETE</u>
ATTRIBUTE_CLASS	N(VERATT)			
DATA_BASE	N(VERDB)			
DATA_FIELD	N(VERDFLD)			
DATA_ITEM	N(VERD1)			
DESC_TEXT	N(VERDSTX) S(WRTDSC4)			S(WRTDSC4)
DESCRIPTION_TYPE	N(VERDSTP)			
DOMAIN_CLASS	N(VERDOM)			
ENTITY_CLASS	N(VERENT)			
KEYWORD	N(VERKW)			
MODEL_CLASS	N(VERMOD)			
RECORD_SET	N(VERRSET)			
RECORD_TYPE	N(VERRT)			

3.2.8.13 COPY ENTITY - Copy a conceptual entity class.

A. Function:

COPY ENTITY allows the NDDL user to perform the following:

1. copy an entity class within the same model, giving it a different name;
2. copy, at the User's discretion, the descriptions, aliases, and keywords for the entity.
3. copy an entity from one model to another, generating the NDDL commands that the user would otherwise have to type in.
4. an intra-model copy entity will allow the user to generate the NDDL commands to create copies of all subordinate entity classes, using the existing definitions in the original model.
5. an intra-model copy entity will alternatively allow the user to generate the NDDL commands to create copies of all associated relation classes of the subject entity.

B. CDM Requirements:

The entity to be copied must previously exist in the CDM. The entity copied to must not exist.

C. Processing:

1. Begin processing by determining if the copy is inter- or intra-model. If a model name is specified, assume intra-model without checking against the current model. This will allow the user to use COPY ENTITY to generate NDDL for the current model. The from-model is verified. The current model is assumed for the to-model. Also the intra-model must specify a new entity name. The from and to entity names are verified. Finally, perform step 3, to perform the copy directly if an intra-model copy was determined or an inter-model copy without the selection of STRUCTURE or RELATIONS. Else step 3 to generate

NDDL commands shall be performed.

2. For an internal copy entity, the database is updated directly.
  - 2.1 A new unique number is obtained and the entity and its primary name stored. The "TO" entity name will be made primary.
  - 2.2 If the user desired, all keywords for the old entity are associated with the new entity.
  - 2.3 If the user desired, all descriptions from the old entity are copied to the new entity.
  - 2.4 If the user desired, all aliases for the old entity are copied.
  - 2.5 If the copy entity was intra-model, no attributes can be copied since they are already owned (by the "FROM" entity). If not, the attributes will be copied:
    - 2.5.1 For each attribute owned by the "FROM" entity:
      - 2.5.1.1 The attribute is checked in the "TO" model. If not found, obtain a new number and store the new attribute and its name as primary.
      - 2.5.1.2 If the user desired, all aliases for the original attribute all copied to the new attribute.
      - 2.5.1.3 If the user desired, all keywords for the original attribute are copied to the new attribute.
      - 2.5.1.4 If the user desired, all descriptions for the original attribute are copied to the new attribute.
      - 2.5.1.5 The newly created attribute can be stored as owned attributes and attribute use classes for the new entity.



- 2.5.1.6 All key classes for the "FROM" entity can be copied to the new entity. The key classes of the attribute being copied, found in step 2.5.1, are stored in a table. If not already in the table, a new key class is established with the new attribute as its key class member. If the key class was on the list, a new key class member is created for the new attribute and the new key class previously created.
3. When generating NDDL to perform the COPY ENTITY, determine if the user requested "WITH STRUCTURE" or "WITH RELATIONS".
    - 3.1 For COPY with STRUCTURE, generate the NDDL to copy the entity itself (the top node). Then:
      - 3.1.1 Generate the Create Attribute commands for each attribute owned by the entity not already found in the "TO" model. Finally, generate NDDL to make the attribute owned by the new entity. Attribute Descriptions are "copied" by generating DESCRIBE commands and attribute aliases are "copied" by generating CREATE ALIAS commands. When copying aliases the target model is checked to insure the name has not been used before.
      - 3.1.2 If the user desired, keywords for the entity are "copied" by adding a keyword clause to the command.
      - 3.1.3. If the user desired, aliases are copied for the entity by generating CREATE ALIAS commands.
      - 3.1.4 If the user desired, all descriptions for the entity are copied by generating DESCRIBE commands.
      - 3.1.5 The key classes for the top node of the structure can be generated next. A search of the key classes and members is made of the original entity and stored in a structure. The original ED\_NO, KC\_NO, KC\_NAME, KCM\_TAG\_NO and KCM\_TAG\_NAME are

saved.

- 3.1.6 An Alter Entity command for the new top node can now be generated which declares each of the key classes and its members found in the data structure stored in 3.1.5.
- 3.1.7 All other attributes associated with the hierarchial structure of entities below the original top node are "copied" through generation of CREATE ATTRIBUTE commands as described in 3.2.1. The search controlling this generation makes use of a recursive search of the CDM's RELATION\_CLASS table.
- 3.1.8 All other entities of the hierarchical structure below the top node are "copied" by generating the appropriate commands as was done in step 3.1.2 through 3.1.6. A recursive search similar to the one of step 3.1.7 is done. For each primary entity name identified, its name and number are stored in a table.
  - 3.1.8.1 The owned attributes are determined and associated with the new entity by generating the "owns" clause.
  - 3.1.8.2 If the user wishes keywords copied, the "keyword" clause is also generated.
  - 3.1.8.3 For the alias entity names found in the search of 3.1.8 and if the user wishes aliases copied, then CREATE ALIAS commands for the new entity are generated.
  - 3.1.8.4 Finally, if the user desires, all descriptions for the new entity are copied by generating DESCRIBE commands.
- 3.1.9 Now that all entities have been created, the Relation Classes connecting the structure can be generated. This must be done in a top down manner, migrating key classes. After all attributes are migrated into an entity, an ALTER Entity can be generated to create the new key classes in preparation

for key migration to the next level down.

- 3.1.9.1 A list of all key classes and their members is created by using a recursive search of the CDM Table `RELATION_CLASS`.
- 3.1.9.2 A search of the CDM is made for all relation classes "below" the top node in the original model. The results of this search are sorted by level, relation class number and dependent entity. Thus all create `RELATIONS` for one level can be done at a time, allowing only one `ALTER ENTITY` per new entity and insuring that all attributes are declared as key before being migrated.
- 3.1.9.3 A search is made of the CDM's `COMPLETE_RELATION` table. The key class number, if found, is used to search the table built in 3.1.9.1 and a `MIGRATES` clause can be generated for the `CREATE RELATION` command. The `SET` clause is generated and for each attribute which is in the key class, its original name (independent) and new name (dependent) are explicitly generated.
- 3.1.9.4 If the user desired, keywords for the relation are "copied".
- 3.1.9.5 If the user desired, descriptions for the relation are copied. Aliases for relations are not supported by the CDM.
- 3.1.9.6 Finally, the `ALTER_ENTITY` is generated to assign its key classes, like step 3.1.5 and 3.1.6.
- 3.2 For `COPY` with `RELATIONS`, the entity itself to be copied is generated as was done in step 3.1.1 through 3.1.6.
  - 3.2.1 The entities and the relations one level below the entity being copied are also generated as done in 3.2 above.

DS 620141100  
1 November 1985

3.2.2 The key classes are generated for each entity copied.

3.2.3 Similar to 3.2.1, entities and relations one level above the entity being copied are also generated as done in 3.2.

DS 620141100  
1 November 1985

CDM TABLES ACCESSED - COPY ENTITY

(S=SQL N=NDML)

<u>TABLE NAME</u>	<u>SELECT</u>	<u>MODIFY</u>	<u>INSERT</u>	<u>DELETE</u>
AC_KEYWORD	N(GENAKW)			
ATTRIBUTE_CLASS	N(VERATT)			
ATTRIBUTE_CLASS	N(CMBOA)			S(INSAC)
ATTRIBUTE_USE_CL	N(BLKCL1)			S(INSauc)
ATTRIBUTE_CLASS	N(COPYAC)			
ATTRIBUTE_USE_CL	N(COPYAC)			
ATTRIBUTE_CLASS	S(DEPATT)			
ATTRIBUTE_USE_CL	S(DPKCLST)			
ATTRIBUTE_CLASS	N(GENOA)			
ATTRIBUTE_USE_CL	S(SELIAUC)			
ATTRIBUTE_USE_CL	N(SELIKEY)			
ATTRIBUTE_NAME	N(CMBACAL)			
ATTRIBUTE_NAME	N(CMBOA)			
ATTRIBUTE_NAME	S(DEPATT)			
ATTRIBUTE_NAME	N(GENOA)			
ATTRIBUTE_NAME	N(VERATT)			
COMPLETE_RELATION	N(VERRCC)			
DESC_TEXT	N(GENDESC)			S(WRTDESC)
DOMAIN_CLASS	N(DMBOA)			
DOMAIN_CLASS	S(DEPATT)			
EC_KEYWORD	N(CMBEKW)			
EC_KEYWORD	N(GENEKW)			
EC_KEYWORD	N(VERKWE)			
EC_KEYWORD	N(WRTECKW)			
ENTITY_CLASS	N(BLKCL1)			
ENTITY_CLASS	N(CMBALI)			
ENTITY_CLASS	N(COPYAC)			
ENTITY_CLASS	S(SELIAUC)			

DS 620141100  
1 November 1985

CDM TABLES ACCESSED - COPY ENTITY

(S-SQL N-NDML)

<u>TABLE NAME</u>	<u>SELECT</u>	<u>MODIFY</u>	<u>INSERT</u>	<u>DELETE</u>
ENTITY_NAME	N(CMBALI)			(INSECNM)
ENTITY_NAME	S(DEPENT)			
ENTITY_NAME	N(SELECNM)			
ENTITY_NAME	N(WRTENAM)			S(INSKWEC)
INHERITED_ATT_USE	S(SELIAUC)			
INHERITED_ATT_USE	N(SELIKEY)			
KEY_CLASS	N(BLKCL1)			S(INSKC)
KEY_CLASS	S(DPKCLST)			
KEY_CLASS	N(KEYLOOK)			
KEY_CLASS_MEMBER	N(BLKCL1)			S(INSKCM)
KEY_CLASS_MEMBER	S(DPKCLST)			
KEY_CLASS_MEMBER	N(KEYLOOK)			
KEY_CLASS_MEMBER	N(SELIKEY)			
KEYWORD	N(CMBEKW)			
KEYWORD	N(CMBRKW)			
KEYWORD	N(GENEKW)			
KEYWORD	N(GENRKW)			
KEYWORD	N(VERKWE)			
KEYWORD	N(VERKWR)			
KEYWORD	N(GENAKW)			
MODEL_CLASS	N(VERMOD)			
OWNED_ATTRIBUTE	N(CMBOA)			S(INSOAC)
OWNED_ATTRIBUTE	N(COPYAC)			
OWNED_ATTRIBUTE	S(DEPATT)			
OWNED_ATTRIBUTE	N(GENOA)			
RC_KEYWORD	N(CMBRKW)			
RC_KEYWORD	N(VERKWR)			
RELATION_CLASS	S(BLRCKW1)			
RELATION_CLASS	S(DEPATT)			
RELATION_CLASS	S(DEPENT)			
RELATION_CLASS	N(DEPFROM)			
RELATION_CLASS	S(DEPREL)			
RELATION_CLASS	S(DPKCLST)			
RELATION_CLASS	N(INDFROM)			
RELATION_CLASS	N(VERRC)			

3.2.8.14 COPY MODEL - Generate NDDL commands on a user defined file to make a copy of an existing model.

A. Function:

Copy Model performs the following functions:

1. create a new model containing all the entities, owned attributes, inherited attributes, key classes, key class members, relations, complete relations, aliases, keywords and descriptions of the model being copied;
2. generate NDDL commands in a user defined file to copy a model in proper sequence i.e. by level.

B. CDM Requirements:

1. The model to be copied must exist in the CDM.
2. The model to be copied into should not exist.

C. Processing:

1. If the model to copied (the from-model) is not specified, the model name defaults to the current model. The copy model command verifies that the from-model exists, and that the new model to be created (the to-model) does not exist. Processing halts if any of the verification checks fail.
2. NDDL commands to copy the from-model are generated on a user defined file. If the named file does not exist, a new file is created and opened. If the file does exist, the generated commands are appended to the file.
3. First, create all the attributes contained in the from-model, along with their associated keywords, aliases and description text.
4. Then, create all the entity classes contained in the from-model along with associated owned attributes, keywords, aliases, and description text.
5. Build a temporary key list to store all the key

classes and key class members for each entity. This list will be used later to define the key for the entity after all the migrations have been determined for this entity.

6. The ORACLE tree search feature is used to identify inheritance at all lower levels, after the top node is identified. The model being copied must not contain any dependency loops. Generate an Alter Entity add key... command for the top node in this model's tree structure.
7. Another temporary list is built to store all the key classes of the dependent entity which contain attributes inherited via the relation for each level of relations in the from-model. Both these lists are used to generate NDDL statements necessary to create the relations. For each level, migrate the keys, and add keys for each dependant entity in the from-model. NDDL commands are also generated for keywords and descriptions associated with the relation classes.
8. Finally, the user defined file is closed.



<u>CDM TABLES ACCESSED - COPY MODEL</u>		<u>(S=SQL, N=NDML)</u>		
<u>TABLE NAME</u>	<u>SELECT</u>	<u>MODIFY</u>	<u>INSERT</u>	<u>DELETE</u>
AC-KEYWORD	N(GENAKW)			
ATTRIBUTE_CLASS	N(GENOA) S(ALLATT)			
ATTRIBUTE_NAME	N(GENOA) S(ALLATT)			
ATTRIBUTE_USE_CL	N(BLKCLST) N(SELIKEY)			
COMPLETE_RELATION	N(VERRC)			
DESC_TEXT	N(GENDESC)			
DOMAIN_CLASS	S(ALLATT)			
EC_KEYWORD	N(GENEKW)			
ENTITY_CLASS	N(BLKCLST) S(ALLENT) S(TOPNODE) S(ALLREL) S(BLRCKC) S(SELIAUC)			
ENTITY_NAME	S(ALLENT)			
INHERITED_ATT_USE	N(SELIKEY) S(SELIAUC)			
KEY_CLASS	N(BLKCLST)			
KEY_CLASS_MEMBER	N(BLKCLST) N(SELIKEY)			
KEYWORD	N(GENAKW) N(GENEKW) N(GENRKW)			
MODEL_CLASS	S(TOPNODE)			

DS 620141100  
1 November 1985

CDM TABLES ACCESSED - COPY MODEL

(S=SQL, N=NDML)

<u>TABLE NAME</u>	<u>SELECT</u>	<u>MODIFY</u>	<u>INSERT</u>	<u>DELETE</u>
	S(ALLREL) S(BLRCKC)			
OWNED_ATTRIBUTE	N(GENOA)			
RC_KEYWORD	N(GENRKW)			
RELATION_CLASS	S(TOPNODE) S(ALLREL) S(BLRCKC)			

3.2.8.15 CREATE ALIAS - Create an alias for an entity or attribute

A. Function:

Create Alias performs the following function:

1. allows the user to add a secondary name, or alias, for any attribute or entity of the current model.

B. CDM Requirements:

The entity or attribute named in the command must exist in the user's current model.

C. Processing:

1. CREATE ALIAS shall access the object type from the user command. It must be either ATTRIBUTE or ENTITY. The command processor must then access the user entered identifier for the entity attribute and verify its presence with a database lookup. The potential new alias name is also verified to make sure it has not already been used. Finally, having the entity attribute number from the first verification, the new alias name can be inserted.

DS 620141100  
1 November 1985

CDM TABLES ACCESSED - CREATE ALIAS

(S=SQL,N=NDML)

<u>TABLE NAME</u>	<u>SELECT</u>	<u>MODIFY</u>	<u>INSERT</u>	<u>DELETE</u>
ATTRIBUTE_CLASS	N(VERATT)			
ATTRIBUTE_NAME	N(VERATT)		S(INSACNM)	
ENTITY_CLASS	N(VERENT)			
ENTITY_NAME	N(VERENT)		S(INSECNM)	

3.2.8.16 CREATE ATTRIBUTE - Create a conceptual attribute

A. Function:

Create Attribute performs the following functions:

1. create an attribute for a model;
2. assign a domain for the attribute;
3. add keywords to an attribute.

B. CDM Requirements:

A current model must be established.

C. Processing:

1. If a domain is specified, the Create Attribute command verifies the existence of the domain to which the attribute is being assigned. If the domain is not specified, the domain will default to zero or undefined.
2. Next, a check is performed to verify that the attribute does not previously exist in the model. Then the attribute is inserted into the ATTRIBUTE\_CLASS and ATTRIBUTE\_NAME tables. This attribute is created as the Primary attribute.
3. If keywords are to be added to the attribute, verify that the keyword has not previously been assigned to the attribute. Keyword references are then created for the attribute by inserting into the AC\_KEYWORD table. Also, the keyword will be inserted into the KEYWORD table if it did not previously exist.
4. Processing halts if any of the verification checks fail.

DS 620141100  
1 November 1985

CDM TABLES ACCESSED - CREATE ATTRIBUTE

(S-SQL, N-NDML)

<u>TABLE NAME</u>	<u>SELECT</u>	<u>MODIFY</u>	<u>INSERT</u>	<u>DELETE</u>
AC_KEYWORD	N(ADDKVA)		S(INSKWAC)	
ATTRIBUTE_CLASS	N(VERATT)		S(INSAC)	
ATTRIBUTE_NAME	N(VERATT)		S(INSACNM)	
DOMAIN_CLASS	N(VERDOM)			
EC_KEYWORD	N(ADDKWE)		S(INSKVEC)	
KEYWORD	N(VERKW)		S(INSKW)	
RC_KEYWORD	N(ADDKWR)		S(INSKWRC)	

3.2.8.17 CREATE DOMAIN - Create a domain

A Function:

Create Domain performs the following functions:

- 1 adds a new domain to the system.
- 2 associates a standard data type with the new domain.
- 3 associates optional user-defined data types with the new domain

B CDM Requirements

- 1 The domain must not previously exist in the system
- 2 The standard data type for the new domain must be specified in the first data type clause
- 3 Additional data types may be specified in subsequent data type clauses. These data-types are considered user-defined data types for the new domain

C Processing

- 1 Create Domain verifies that the domain to be added does not exist in the system. Then the new domain is inserted into the DOMAIN CLASS table
- 2 The data type specified for the standard or user defined DATA TYPE NAME is verified as a legal data type, and the number of decimals, if specified, must not exceed the maximum field length of DATA TYPE NAME. The DATA TYPE NAME is then inserted into USER DEF DATA TYPE tables as a standard or user defined data type for the new domain

DS 620141100  
1 November 1985

CDM TABLES ACCESSED - CREATE DOMAIN (S-SQL,N-NDML)

<u>TABLE NAME</u>	<u>SELECT</u>	<u>MODIFY</u>	<u>INSERT</u>	<u>DELETE</u>
DOMAIN_CLASS	N(VERDOM)		S(INSDOM)	
USER_DEF_DATA_TYPE	N(VERSDT) N(VERDT) N(VERDTD)		S(INSDT)	
DATA_TYPE	N(VERTYP)			



3.2.8.18 CREATE ENTITY - Create a Conceptual Entity Class

A. Function:

CREATE ENTITY will allow the MDDL user to perform the following functions:

1. create a new entity class for the current model;
2. associate attributes as owned attributes for this entity;
3. define key classes of the owned attributes only. (unowned attributes must be migrated to the entity by use of the CREATE or ALTER RELATION commands;
4. associate keywords with the entity.

B. CDM Requirements:

The entity being created must not previously be found in the current model. Any attribute referenced must already be defined for the current model and must not be previously owned by any entity.

C. Processing:

1. First the entity itself is created. The name is accessed from the command and database lookup verifies that it is not already present. If not found, the entity class is assigned a unique number and stored and the user entered name is stored as the entity's primary or preferred name, not an alias.
2. Any user specified attributes are then processed. Each attribute name must be found in the current model and also verified not to be owned by any other entity in the model. If these conditions are met, the attribute can be associated with the entity by storing an occurrence of OWNED ATTRIBUTE. Finally, a unique tag number is obtained, and the attribute is also stored as an ATTRIBUTE\_USE CLASS of this entity.
3. When the user has specified any key classes to be defined for this entity, the user may omit the

attribute names for the key class. Therefore, the key class name may be taken as the attribute name if none follows the - sign. For each key class specified the program must:

- 3.1 Verify the key class name was not used for this entity.
- 3.2 Obtain a unique number for the key class.
- 3.3 Store a key class occurrence for the entity first created.
- 3.4 Process each attribute name specified for the current key class.
  - 3.4.1 Determine if the attribute has already been associated with the entity as an ATTRIBUTE USE CLASS.
  - 3.4.2 If it had not, an attempt is made to make the attribute owned by the entity as specified in step 2 above.
  - 3.4.3 Finally, an occurrence of KEY CLASS MEMBER can be stored.
- 4 Finally, any keywords specified by the user can be related to the entity first created. For each keyword entered.
  - 4.1 The keyword is verified in the table of keywords (independent of node)
  - 4.2 If the keyword is new, then a unique number for the keyword is obtained and a keyword occurrence is stored
  - 4.3 Finally, the keyword to entity cross reference is (EC KEYWORD) stored, relating the keyword to the entity just created

DS 620141100  
1 November 1985

CDM TABLES ACCESSED - CREATE ENTITY

(S-SQL, N-NDML)

<u>TABLE NAME</u>	<u>SELECT</u>	<u>MODIFY</u>	<u>INSERT</u>	<u>DELETE</u>
ATTRIBUTE CLASS	N(VERATT)			
ATTRIBUTE NAME	N(VERATT)			
ATTRIBUTE USE CL	N(VERAUC)		S(INSAUC)	
EC KEYWORD	N(ADOKVE)		S(INSKVEC)	
ENTITY CLASS	N(VERENTC)		S(INSEC)	
ENTITY NAME	N(VERENT)		S(INSECHN)	
KEYWORD	N(VERKV)		S(INSKV)	
KEY CLASS	N(VEREC)		S(INSEC)	
KEY CLASS MEMBER			S(INSECH)	
OWNED ATTRIBUTE	N(VEROAC)		S(INBOAC)	

### 3.2.8.19 CREATE MAP - Create CS-IS mapping

#### A. Function:

The CREATE MAP command allows the user to map an attribute use class (AUC) to a data field or set, or from a RELATION CLASS to a set.

#### B. CDM Requirements:

The AUC or the relation class from which you map and any or all of the following which are to be mapped to: the database, record name, data field name, data type name, set name and member record name; must all exist in the CDM.

#### C. Processing:

Processing differs depending on the type of mapping attempted.

- 1 For an AUC-to-data field mapping, the following rules apply:
  - 1 1 The AUC must not have previously been mapped to a set
  - 1 2 The AUC must not have previously been mapped to a data field
  - 1 3 If a data type name is not entered, the standard data type for the AUC's domain is used
  - 1 4 Only one primary mapping may exist for an AUC
  - 1 5 Multiple secondary mappings may exist if there is a pre existing primary mapping
  - 1 6 If the primary or secondary mapping is not specified, the default is primary

If all rules are obeyed, a PROJECT DATA FIELD entity is inserted into the CDM
- 2 For an AUC to set mapping, the following rules apply

- 2.1 A data field map must not exist for the AUC.
- 2.2 The set to be mapped to must have a single record type for its members.
- 2.3 The set to be mapped to must not previously have been mapped from a relation class or another AUC.
- 2.4 All AUC to set maps must map to the same database for a particular AUC.
- 2.5 All AUC to set maps must contain a value which must be unique for a particular AUC.
- 2.6 All mapped to sets from a AUC must have the same record type as its owner.

If all rules are obeyed, an AUC\_ST\_MAPPING entity is inserted into the CDM.

3. For a relation-class-to-set-mapping, the following rules apply:
  - 3.1 There must be no previous mappings to the set.
  - 3.2 The member record name is required if the set mapped to contains member records of more than one type.

If all rules are obeyed, an RC\_BASED\_REC\_SET entity is inserted into the CDM.

4. A relation class to data field mapping is meaningless and therefore illegal.

DS 620141100  
1 November 1985

CDM TABLES ACCESSED - CREATE MAP

(S=SQL, N=NDML)

<u>TABLE NAME</u>	<u>SELECT</u>	<u>MODIFY</u>	<u>INSERT</u>	<u>DELETE</u>
ATTRIBUTE_CLASS	N(FINDDOM)			
ATTRIBUTE_USE_CL	N(VERAUC)			
AUC_ST_MAPPING	N(VOMAPS) N(FNDASA) N(VERASH)		S(INSAUCS)	
DATA_BASE	N(VERDB) N(VERDF)			
DATA_FIELD	N(VERDF)			
ENTITY_CLASS	N(VERENT)			
PROJECT_DATA_FIELD	N(PDFSRCH) N(VERPDF)		S(INSPDF)	
RC_BASED_REC_SET	N(VERRCBS) N(VERRCMP)		S(INSRCRS)	
RECORD_SET	N(VOMAPS) N(VERSMS)			
RECORD_TYPE	N(VERDF)			
RELATION_CLASS	N(VERRC)			
SET_TYPE_MEMBER	N(FND1MEM)			
USER_DEF_DATA TYPE	N(VERSDT) N(VERDTD)			

3.2.8.20 CREATE MODEL - Create a new IDEF1 model.

A. Function:

A new model is created as unchecked in the system.

B. CDM Requirements:

The model to be created must exist in the CDM.

C. Processing:

1. First, verify whether the model to be created exists in the system. If it does, flag an error; otherwise, obtain a model number for the model name.
2. Store the model number, model name into the CDM.

DS 620141100  
1 November 1985

CDM TABLES ACCESSED - CREATE MODEL

(S=SQL, N=NDML)

<u>TABLE NAME</u>	<u>SELECT</u>	<u>MODIFY</u>	<u>INSERT</u>	<u>DELETE</u>
MODEL_CLASS	N(VERMOD)			S(INSMOD)



3.2.8.21 CREATE RELATION - Create a relation class

A. Function:

Create Relation performs the following functions:

1. create a relation class for user entered independent and dependent entity classes;
2. create associated keywords for the relation class;
3. migrate a key class from the independent entity class to the dependent entity class.

B. CDM Requirements:

The independent and dependent entity class must exist in the current model. If the migrates clause is present, the key class for the independent entity must exist in the current model.

C. Processing:

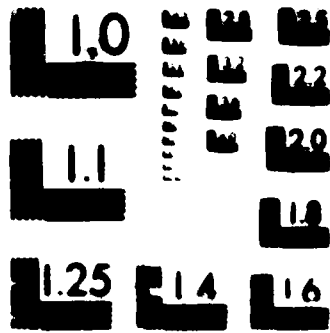
1. The Create Relation process verifies that both the independent and dependent entity class exist in the current model. If both do not exist, an error is issued and processing is terminated. A check is made to determine if the relation class to be created already exists between the user entered entity classes. If one does exist, as above, an error is issued and processing is terminated.
2. Next, the process validates the cardinality of the relation class to be created. If a cardinality is omitted by the user, the relation is assigned a default value. The default for the independent cardinality is a value of one. The default for the dependent cardinality is a value of zero for the left dependent cardinality and a value of 99 for the right dependent cardinality.
3. If the migrates clause is entered, the existence of the key class for the independent entity is determined. If the key class does not exist, an error is issued and processing is terminated. An attribute use class and an inherited attribute use class for the dependent entity class is created for

DS 620141100  
1 November 1985

each key class member of the independent entity class migrated to the dependent entity class. If the set phrase is specified, TAG\_NAME2 (the independent entities tag name) is migrated with the new name of TAG\_NAME1.

4. If a keyword is to be added to the relation class, the keyword table is searched to determine whether the keyword exists. If it does not exist, the new keyword is inserted into the keyword table. The new keyword is then associated with the relation class.





U.S. GOVERNMENT PRINTING OFFICE: 1963 O - 354-100

DS 620141100  
1 November 1985

CDM TABLES ACCESSED - CREATE RELATION (S-SQL, N-NDML)

<u>TABLE NAME</u>	<u>SELECT</u>	<u>MODIFY</u>	<u>INSERT</u>	<u>DELETE</u>
ATTRIBUTE USE CL	N(VERAUC) N(ADDNIG)		S(INSBAC)	
COMPLETE RELATION	N(VERBOC)		S(INSBRC)	
INHERITED ATT USE			S(INSIAUC)	
KEY CLASS	N(VERKC)			
KEY CLASS MEMBER	N(ADDNIG)			
KEYWORD			S(INSKV)	
RC KEYWORD	N(ADDKVR)		S(INSKVRC)	
RELATION CLASS	N(VERBC)		S(INSRC)	

**3.2.8.22 CREATE VIEW - Create an external schema view and mappings to the conceptual schema.**

**A. Function:**

Create a view of the entity class and relation class existing in the conceptual schema of the Common Data Model.

**B. CDM Requirements:**

The following elements must exist in the CDM:

1. independent entities specified;
2. dependent entities specified;
3. relation classes named;
4. entity classes of the view;
5. data items defined and attribute use class of the entities must be from the same domain.

**C. Processing:**

1. Verify if the view to be created already exists. If it does, flag a user error.
2. Obtain a unique number for the view to be created.
3. Insert the view name and view number into the CDM.
4. Construct an in-code tables from the user information in the create view command syntax.
5. Examine the from in-code table, if empty, indicates only one entity class selected. Fill in with the first entity name from the select clause. If an \* was entered in the select clause, its an error since there is no way of knowing what the entity class should be.
6. If only one entity class has been entered in the select clause, the following functions must be performed:

- 6.1 Verify entity.
  - 6.2 Verify that data items and tags are from the same domain class.
  - 6.3 Insert view number and tag name into DATA\_ITEM and PROJECT\_DATA\_ITEM, if the data item list is blank.
7. If multiple entity class is in the select clause, the following functions must be performed:
- 7.1 Expand abbreviations to entity class names in select and where clauses.
  - 7.2 Verify each relation in the where clause.
  - 7.3 Verify entity and domain.
  - 7.4 Store SEC\_RC.

DS 620141100  
1 November 1985

CDM TABLES ACCESSED - CREATE VIEW

(S-SQL, N-NDML)

<u>TABLE NAME</u>	<u>SELECT</u>	<u>MODIFY</u>	<u>INSERT</u>	<u>DELETE</u>
ATTRIBUTE_CLASS	N(ALLVIEW) N(GETDOM)			
ATTRIBUTE_USE_CL	N(ALLVIEW) N(GETDOM)			
DATA_ITEM			S(INSDI)	
ENTITY_CLASS	N(VERENT)			
PROJECT_DATA_ITEM			S(INSPI)	
RELATION_CLASS	N(VERRC)			
SEC	N(VERVIEW)		S(INSSEC)	
SEC_RC_COMPONENT			S(INSSECR)	
USER_DEF_DATA_TYPE	N(VERSDT) N(VERDTD)			



**3.2.8.23 DEFINE DATABASE - Describe the definition of a database to the CDM internal schema.**

**A. Function:**

The command defines a database to the CDM.

**B. CDM Requirements:**

The database to be defined must not exist in the CDM.

**C. Processing:**

1. Verify the existence of the database to be defined if it exists flag a user error.
2. Obtain a unique number for the database.
3. Insert the database entity occurrence.
4. If DBMS is ORACLE:
  - 4.1 Insert the password entity occurrence of the database.
5. If DBMS is TOTAL:
  - 5.1 Verify the existence of the area of the database. If it already exists, flag a user error.
  - 5.2 Insert the area entity occurrence of the database in the CDM.
6. If the DBMS is IMS:
  - 6.1 Check if the start position and feedback length are provided in the command.
  - 6.2 If they are not in the command, flag a user error.
  - 6.3 Verify the existence of the PSB of the IMS database. If it already exists, flag a user error; otherwise, insert the PSB entity occurrence and the PCB entity occurrence.

DS 620141100  
1 November 1985

7. If DEMS is CODASYL:
  - 7.1 Insert the schema occurrence of the CODASYL database.
  - 7.2 Verify the existence of the area of the CODASYL database. If it already exists, flag a user error. Otherwise, insert the area occurrence.

DS 620141100  
1 November 1985

<u>CDM TABLES ACCESSED - DEFINE DATABASE</u>					(S=SQL,N=NDML)
<u>TABLE NAME</u>	<u>SELECT</u>	<u>MODIFY</u>	<u>INSERT</u>	<u>DELETE</u>	
DB_PASSWORD			S(INSPWRD)		
DATA_BASE	N(VERDB)			S(INSDB)	
DATA_BASE_AREA	N(VERAREA)		S(INSAREA)		
PSB	N(VERPSB)		S(INSPSB)		
PSB_PC			S(INSPCB)		
REUSABLE_NUMBER	S(NRGET)				
SCHEMA_NAMES			S(INSSDT)		

**3.2.8.24 DEFINE RECORD - Creates a Record Type/Table/Segment for a previously defined database/PCB**

**A. Function:**

Define Record performs the following functions:

1. verifies that the database exists;
2. verifies that the record has not previously been defined;
3. inserts the record into the CDM database;
4. if the database is CODASYL, the process verifies that the database area already exists and inserts the record into the CDM database area of assignment;
5. if the DBMS is IMS, the process inserts the IMS\_SEGMENT\_SIZE and the SEGMENT\_DATA\_FIELD;
6. if the DBMS is CODASYL or TOTAL, the command inserts the record key and the record key members.

**B. CDM Requirements:**

This process requires a previously defined database.

**C. Processing:**

1. Define Record/Table/Segment verifies that the database exists in the CDM and that the record has not been previously defined for that database. If the database exists and the record has not been defined, the record is inserted in the CDM database.
2. CODASYL DBMS - The process verifies the database area for the database and then inserts the record into the CDM DB area assignment.
3. IMS DBMS - The process inserts the record segment size in the CDM IMS-Segment size table.
4. ALL DBMS - The process inserts each specified data

DS 620141100  
1 November 1985

field for the record into the CDM data field table. Additionally, for IMS DBMS the data field inserted in the CDM Segment data field.

5. TOTAL & CODASYL DBMS - If record key information has been specified, the process verifies that the record key has not been previously defined.
6. If not, a - key is created for the newly defined record. A check is made to verify that each data field mentioned as a key member has been defined as a data field for this record. Each data field is then created as a record key member.

DS 620141100  
1 November 1985

CDM TABLES ACCESSED - DEFINE RECORD

(S-SQL, N-NDML)

<u>TABLE NAME</u>	<u>SELECT</u>	<u>INSERT</u>	<u>MODIFY</u>	<u>DELETE</u>
DATA_BASE	N(VERDBAS)			
DATA_BASE_AREA	N(VERAREA)	S(INSDAA)		
DATA_FIELD		S(INSDFLD)		
IMS_SEGMENT_SIZE		S(INSISS)		
RECORD_KEY	N(VERRK)			
RECORD_KEY		S(INSRKEY)		
RECORD_KEY_MEMBER	N(VERRKM)	S(INSRKM)		
RECORD_TYPE	N(VERRT)	S(INSRTYP)		
SEGMENT_DATA_FIELD		S(INSSDFL)		

**3.2.8.25 DEFINE SET - define or internal set/path for CODASYL, TOTAL and IMS DBMS's**

**A. Function:**

Define set performs the following functions:

1. create a set/path for a CODASYL (VAX-11, IDMS, IDS), IMS or TOTAL DBMS;
2. allow a set between owner and multiple members for CODASYL, but only single member for other DBMS.

**B. CDM Requirements:**

The database must be established during the session. The owner and member record types must exist. If creating a set for a TOTAL DBMS, the data field from the variable record must exist.

**C. Processing:**

1. Define Set verifies the existence of the database/PCB in which the set is to be created. If the database is not specified, it defaults to the database established during the current session.
2. Next, a check is performed to verify that the set to be created does not exist. For an IMS database, the path name is derived by combining the owner record and member record names.
3. For a TOTAL or IMS database, verify that the owner and member records have previously been defined. In addition, verify that the data field of the variable (member) record to which the set is to be linked, has also been defined. The set information is then inserted into the DF\_SET\_LINKAGE, SET\_TYPE\_MEMBER and RECORD\_SET tables.
4. For a CODASYL DBMS multiple members are allowed. Verify that the owner record and member record(s) have been previously defined. A required/optional entry must be specified for the member record types. The set information is then inserted into the SET\_TYPE\_MEMBER and RECORD\_SET tables.

DS 620141100  
1 November 1985

5. Processing halts if any of the verification checks fail.



DS 620141100  
1 November 1985

CDM TABLES ACCESSED - DEFINE SET

(S-SQL, N-NDML)

<u>TABLE NAME</u>	<u>SELECT</u>	<u>MODIFY</u>	<u>INSERT</u>	<u>DELETE</u>
DATA_BASE	N(VERDBAS)			
DATA_FIELD	N(VERDFLD)			
DF_SET_LINKAGE			S(INSDSL)	
RECORD_SET	N(VERRSET)		S(INSRSET)	
RECORD_TYPE	N(VERRT)			
SET_TYPE_MEMBER			S(INSSTM)	

3 2 8 25 DESCRIBE Describe Objects

A Function

The DESCRIBE command allows description text of the following object types to be entered, modified or deleted. The object types are

- 1 Database
- 2 Set
- 3 Record
- 4 Data field
- 5 Domain
- 6 User data type
- 7 View
- 8 Data item
- 9 Keyword
- 10 Entity
- 11 Attribute
- 12 Relation

B CDM Requirements

The object to be described must exist in the CDM

C Processing

- 1 The user entered description type is validated against the description type table maintained by the CDM administrator
- 2 The object's existence is validated
- 3 The description text originates from three sources: a text file, from the command line or from the UI UT1 Screen Editor. If the text originated from

a text file and if the file contains data, only pre-existing description text of the proper type is deleted prior to the insertion of the new text. If the file contains no data, the description text is not deleted and an error message is generated. If the text originates from the command line, the old description text, if any, is deleted prior to the insertion of new text, if any. Therefore, to delete old description text, the user must describe the object with a null description on the command line.

- 4 If the description text is to come from the UI/UTI Screen Editor, pre-existing description text if any, is extracted from the database and written to a file
- 5 The UI/UTI Screen Editor is called to edit the file. If changes are made, the old description text is replaced by the text output from the editor. If no editing changes were made, the database is not modified.

DS 620141100  
1 November 1985

CDM TABLES ACCESSED - DESCRIBE

(S=SQL, N=NDML)

<u>TABLE NAME</u>	<u>SELECT</u>	<u>MODIFY</u>	<u>INSERT</u>	<u>DELETE</u>
ATTRIBUTE_CLASS	N(VERATT)			
ATTRIBUTE_NAME	N(VERATT)			
DATA_BASE	N(VERDB)			
DATA_FIELD	N(VERDFLD)			
DATA_ITEM	N(VERDI)			
DESC_TEXT	N(OUTDESC)		S(RDDESC, FILEINS) S(RDDESC, STRINS)	S(DELTXT)
DOMAIN_CLASS	N(VERDOM)			
ENTITY_CLASS	N(VERENT)			
ENTITY_NAME	N(VERENT)			
KEYWORD	N(VERKW)			
RECORD_SET	N(VERRSET)			
RECORD_TYPE	N(VERRT)			
RELATION_CLASS	N(VERRC)			
SEC	N(VERVIEW)			
USER_DEF_DATA_TYPE	N(VERUDTN)			

**3.2.8.27 DROP ALIAS - Deletes an alias established for an attribute or entity name.**

**A. Function:**

Drop Alias performs the following functions:

1. verifies whether the alias is for an attribute or entity;
2. verifies that the alias exists for the attribute or entity for a specified model;
3. deletes the Alias for the attribute or entity.

**B. CDM Requirements:**

Drop Alias requires the presence of an Alias for the attribute or entity.

**C. Processing:**

1. The DROP ALIAS process will determine whether the alias is of an attribute or entity and verify if the attribute or entity exists for the specified model.
2. The process will then verify the Alias name and delete the alias for the attribute/entity from the CDM Entity or Attribute name table.

DS 620141100  
1 November 1985

CDM TABLES ACCESSED - DROP ALIAS

(S-SQL, N-NDML)

<u>TABLE NAME</u>	<u>SELECT</u>	<u>MODIFY</u>	<u>INSERT</u>	<u>DELETE</u>
ATTRIBUTE_CLASS	N(VERATT)			
ATTRIBUTE_NAME	N(VERATT) N(GETACAL)			S(DELACAL) S(DELECAL)
ENTITY_CLASS	N(VERENT)			
ENTITY_NAME	N(VERENT) N(GETECAL)			

**3.2.8.28 DROP ATTRIBUTE - Drop a Conceptual Attribute**

**A. Function:**

Drop Attribute deletes one or more user specified attributes from the CDM.

**B. CDM Requirements:**

The attribute(s) to be dropped must exist in the current model.

**C. Processing:**

1. DRPATT, after verifying that the attribute exists, determines whether the attribute to be dropped is owned. If so, the attribute is deleted from the OWNED\_ATTRIBUTE and ATTRIBUTE\_USE\_CL tables.
2. If the attribute to be dropped has been inherited, all instances of inheritance are deleted. The tables affected are INHERITED\_ATT\_USE, ATTRIBUTE\_USE\_CL, BY\_CLASS\_MEMBER and DESC\_TEXT. The ORACLE tree search feature is used to identify inheritance at all lower levels.
3. After all owned or inherited instances of the attribute are deleted, the attribute is deleted from ATTRIBUTE\_CLASS, ATTRIBUTE\_NAME, AC\_KEYWORD and DESC\_TEXT.
4. If the attribute deleted was a by class member, and if it was the only member of a particular by class the corresponding entries in the BY\_CLASS, complete relation and DESC\_TEXT tables are deleted.

DS 620141100  
1 November 1985

CDM TABLES ACCESSED - DRPATT

(S=SQL, N=NDML)

<u>TABLE NAME</u>	<u>SELECT</u>	<u>MODIFY</u>	<u>INSERT</u>	<u>DELETE</u>
AC_KEYWORD				S(delackw)
ATTRIBUTE_CLASS	N(veratt)			S(delac)
ATTRIBUTE_NAME	N(veratt)			S(delacnm)
ATTRIBUTE_USE_CL	N(deoac)			S(deaucl)
COMPLETE_RELATION				S(dekmp)
DESC_TEXT				S(deltext)
ENTITY_CLASS	S(delmtk)			
KEY_CLASS	S(delmtk)			S(delkc)
KEY_CLASS_MEMBER	S(delmtc)			S(delkcm)
INHERITED_ATT_USE	S(delmig)			S(deliac)
OWNED_ATTRIBUTE				S(delowac)



3.2.8.29 DROP DATABASE - Delete a database from the Common Data Model.

A. Function:

Drop Database deletes all references to the database from the Common Data Model.

B. CDM Requirements:

1. The database or IMS PCB must exist in the Common Data Model.
2. The Common Data Model database cannot be dropped.

C. Processing:

After verifying that the database or IMS PCB exists, all references to the database or PCB are deleted from the Common Data Model. If any of the data fields for the database or PCB map to the INTEGRATED\_MODEL, their mapping will be deleted. If the mapping was primary, all secondary mappings, even to other databases, are deleted.

DS 620141100  
1 November 1985

CDM TABLES ACCESSED - DROP DATABASE

(S=SQL, N=NDML)

<u>TABLE NAME</u>	<u>SELECT</u>	<u>MODIFY</u>	<u>INSERT</u>	<u>DELETE</u>
AUC_ST_MAPPIN				S(DELASH1)
DATA_BASE	N(VERDBAS)			S(DELDBS1)
DATA_BASE_AREA				S(DELDBA1)
DATA_FIELD				S(DELDFL1)
DB_AREA_ASSIGNMENT				S(DELDAA1)
DF_SET_LINKAGE				S(DELDSL1)
IMS_SEGMENT_SIZE				S(DELISS1)
PROJECT_DATA_FIELD	N(PDFDB)			S(DELPDFT) S(DEL1PDF)
PCB_PSB				S(DELPCB)
RC_BASED_REC_SET				S(DELRBR1)
RECORD_KEY				S(DELRKY1)
RECORD_KEY_MEMBER				S(DELRKM1)
RECORD_SET	N(VERSTNO)			S(DELRST2)
RECORD_TYPE				S(DELRTY1)
SCHEMA_NAME				S(DELSN1)
SEGMENT_DATA_FIELD				S(DELSDF1)
SET_TYPE_MEMBER				S(DELSTM1)

**3.2.8.30 DROP DOMAIN - Drop a Domain definition from the CDM.**

**A. Function:**

**DROP DOMAIN** allows the NDDL User to drop the definitions of one or more domains from the CDM.

**B. CDM Requirements:**

The domains to be dropped must currently exist, independent of model, and no attributes, data items or data fields must be associated with the data types defined for the domain.

**C. Processing:**

1. For each domain name specified by the user, the following is done:
  - 1.1 The domain name is verified, retrieving its domain number.
  - 1.2 Any attribute class associated with the domain are searched. This search is possible because the standard data type of the domain is the only data type associated with attributes.
  - 1.3 For each data type associated with the domain:
    - 1.3.1 Usage of the data type by any internal schema data fields is determined and displaced to the user.
    - 1.3.2 Usage of the data type by any external schema data items is determined and displaced to the user.
  - 1.4 If the count of usages of any data types of this domain is not zero, the user is given a message and the domain will not be deleted.
  - 1.5 If the domain can be deleted, the following steps are executed:
    - 1.5.1 All data types can be deleted, reassigning their unique numbers to the reusable list. The data type descriptions are also deleted.

DS 620141100  
1 November 1985

1.5.2 The DOMAIN CLASS entry itself can be deleted, along with any associated text descriptions. Its unique number is added to the pool of re-usable numbers.

DS 620141100  
1 November 1985

CDM TABLES ACCESSED - DROP DOMAIN

(S=SQL, N=NDML)

<u>TABLE NAME</u>	<u>SELECT</u>	<u>MODIFY</u>	<u>INSERT</u>	<u>DELETE</u>
ATTRIBUTE_CLASS	N(VERACDT)			
DATA_ITEM	N(VERDIDT)			
DESC_TEXT				S(DELTEXT)
DOMAIN_CLASS	N(VERDOM)			S(DELDOM)
PROJECT_DATA_FIELD	N(VERDFDT)			
USER_DEF_DATA_TYPE	N(DOMUSAG) N(DELDTNO)			S(DELDTD)

**3.2.8.31 DROP ENTITY - Drop a conceptual entity**

**A. Function:**

Drop Entity performs the following functions:

1. delete one or more user specified entities from the CDM.
2. deletes the primary name of the entity and all associated aliases, keywords and description text.
3. deletes all associated owned attributes, attribute use classes and inherited attributes.
4. deletes all associated key classes and key class members.
5. deletes all relation classes associated with the entity and its keywords.

**B. CDM Requirements:**

Each entity to be dropped must exist in the current model.

**C. Processing:**

1. The Drop Entity process verifies that the entity to be dropped exists in the current model. If it does not exist an error is issued and processing is terminated.
2. The process next determines all owned attributes and attribute use classes for the entity and these are dropped. Further, its key classes and attributes inherited via the migrated keys and key class members are also dropped. If the deletion of the entity resulted in any empty key classes for the model, these are then deleted.
3. Next all relations where the entity is independent and dependent are deleted as is its associated keywords.
4. Finally, the primary name of the entity and all of its aliases keywords and description text are

DS 620141100  
1 November 1985

deleted from the model.

DS 620141100  
1 November 1985

CDM TABLES ACCESSED - DROP ENTITY (S-SQL N-MODEL)

<u>TABLE NAME</u>	<u>SELECT</u>	<u>MODIFY</u>	<u>INSERT</u>	<u>DELETE</u>
ATTRIBUTE_USE_CL	N(DELQAC) N(FNDQAC)			S(DELQACL)
COMPLETE_RELATION				S(DELQMPR)
DESC_TEXT				S(DELQTEXT)
EC_KEYWORD				S(DELQCKV)
ENTITY_CLASS	N(VERENT) N(DELQTEC)			S(DELQEC)
ENTITY_NAME	N(VERENT)			S(DELQENM)
INHERITED_ATT_CL	S(DELQIHC)			S(DELQIAC)
KEY_CLASS	S(DELQTEC)			S(DELQKC)
KEY_CLASS_MEMBER	S(DELQTEC)			S(DELQKMT)
OWNED_ATTRIBUTE	N(FNDQAC)			S(DELQOAC)
RC_KEYWORD				S(DELQCKV)
RELATION_CLASS	N(DRPRCE)			S(DELQRC)



**3 2 8 32 DROP FIELD - Drop field from the internal schema.**

**A Function.**

This command deletes all the data fields specified, and its associations and all associated mappings.

**B CDM Requirements.**

The following elements must exist in the CDM.

- 1 Columns/fields/elements/items named on the command.
- 2 Table/record/segment named on the command.
- 3 Database/PCB named on the command.

**C Processing.**

- 1 Verify the existence of the database named. If it does not exist, flag a user error.
- 2 Verify the existence of the record and data field. If it does not exist, flag a user error.
- 3 Add the data field number to the reusable number list
- 4 Delete any textual descriptions of the data field
- 5 Verify the existence of any record sets associated with data field through the DATA\_FIELD\_SET\_LINKAGE entity. For each one found:
  - 5 1 Delete the set from DF SET LINKAGE entity occurrence
  - 5 2 Delete the SET TYPE MEMBER entity occurrences for the record set
  - 5 3 Delete the RECORD SET entity occurrence for this set
  - 5 4 Delete the AUC SET MAPPING entity occurrences for this set
  - 5 5 Delete the RC BASED REC SET entity occurrences

DS 620141100  
1 November 1985

for this set.

- 5.6 Delete the SEGMENT\_DATA\_FIELD entity occurrences for this set.
6. Delete the DATA\_FIELD entity itself.
7. Delete the RECORD\_KEY\_MEMBER entity occurrences for this data field.
8. Delete the RECORD\_KEY entity occurrences for this data field.
9. Delete the PROJECT\_DATA\_FIELD entity occurrences for this data field. If the mappings was primary, delete all other data field mappings for the same attribute use class.

DS 620141100  
1 November 1985

CDM TABLES ACCESSED - DROP FIELD

(S-SQL, N-NDML)

<u>TABLE NAME</u>	<u>SELECT</u>	<u>MODIFY</u>	<u>INSERT</u>	<u>DELETE</u>
AUC_SET_MAPPING				S(DELASM2)
DATA_BASE	N(VERDBAS)			
DATA_FIELD	N(VERDFLD)			
DATA_FIELD				S(DELDL3)
DESC_TEXT				S(DELTEXT)
DF_SET_LINKAGE	N(VERDSL3)			S(DELDL3)
PROJECT_DATA_FIELD	N(PDFDF)			S(DELPDFT) S(DEL1PDF)
RC_BASED_REC_SET RECORD_KEY				S(DELRBR3) S(DELRKY2)
RECORD_KEY_MEMBER				S(DELRKM3)
RECORD_SET				S(DELRST3)
SEGMENT_DATA_FIELD				S(DELSDF3)
SET_TYPE_MEMBER				S(DELSTM3)

**3.2.8.33 DROP KEYWORD - Delete an object keyword.**

**A. Function:**

Drop Keyword performs the following function:

1. delete the named keyword and associations with any attribute, entity and/or relation class.

**B. CDM Requirements:**

The keyword to be dropped must exist in the CDM.

**C. Processing:**

1. Drop keyword verifies the existence of the keyword. The keyword is deleted from the AC\_KEYWORD, EC\_KEYWORD, RC\_KEYWORD, and from the KEYWORD tables.
2. Processing halts if any of the verification checks fail.

DS 620141100  
1 November 1985

CDM TABLES ACCESSED - DROP KEYWORD

(S=SQL, N=NDML)

<u>TABLE NAME</u>	<u>SELECT</u>	<u>MODIFY</u>	<u>INSERT</u>	<u>DELETE</u>
AC_KEYWORD				S(DELKWAC)
DESC_TEXT				S(DELTEXT)
EC_KEYWORD				S(DELKWEC)
KEYWORD	N(VERKW)			S(DELKW)
RC_KEYWORD				S(DELKWRC)

**3.2.8.34 DROP MAP - Delete a CS-IS Mapping**

**A. Function:**

**Drop Map performs the following functions:**

1. delete all mappings from a particular attribute use class (AUC) or from a particular RELATION\_CLASS.

**B. CDM Requirements:**

**The map to be dropped must exist on the CDM.**

**C. Processing:**

1. After validating that the map exists, the appropriate PROJECT\_DATA\_FIELD, AUC\_ST\_MAPPING or RC\_BASED\_REC\_SET entity is deleted.

DS 620141100  
1 November 1985

CDM TABLES ACCESSED - DROP MAP

(S-SQL, N-NDML)

<u>TABLE NAME</u>	<u>SELECT</u>	<u>MODIFY</u>	<u>INSERT</u>	<u>DELETE</u>
ATTRIBUTE_USE_CL	N(VERAUC)			
AUC_ST_MAPPING	N(VERASH)			S(DELASM)
ENTITY_CLASS	N(VERENT)			
ENTITY_NAME	N(VERENT)			
PROJECT_DATA_FIELD	N(PDFSRCH)			S(DELPDFT)
RC_BASED_REC_SET	N(VERRCST)			S(DELRCST)
RELATION_CLASS	N(VERRC)			

**3.2.8.35 DROP MODEL - Delete a model from the CDM.**

**A. Function:**

Drop Model performs the following functions:

1. drop all entities associated with the model.
2. drop all attributes, attribute use classes, and inherited attributes associated with the model.
3. drop all key classes, and key class members associated with the model.
4. drop all relations associated with the model.
5. drop all descriptions, aliases and keywords for the entities, attributes and relations associated with the model.

**B. CDM Requirements:**

The model to be dropped must exist in the CDM.

**C. Processing:**

1. Drop Model verifies that the model to be dropped exists. The INTEGRATED\_MODEL cannot be dropped.
2. For each entity found in the model, its owned attributes, keywords, descriptions and the entity itself is dropped. Further, its key classes and attributes inherited via the migrated keys and key class members are also dropped. Relations where the entity is both dependent and independent are deleted, so also its associated keywords and descriptions.
3. Finally, for each attribute in the model, the attribute keywords, descriptions and the attribute itself is dropped.
4. Processing halts if any of the verification checks fail.



DS 620141100  
1 November 1985

CDM TABLES ACCESSED - DROP MODEL

(S=SQL, N=NDML)

<u>TABLE NAME</u>	<u>SELECT</u>	<u>MODIFY</u>	<u>INSERT</u>	<u>DELETE</u>
AC_KEYWORD				S(DELACKW)
ATTRIBUTE_CLASS	N(FNDACH)			S(DELAC)
ATTRIBUTE_NAME	N(SELACNM)			S(DELACNM)
ATTRIBUTE_USE_CL	N(DLMDAUC)			S(DELAUCL)
COMPLETE_RELATION				S(DELCMPR)
DESC_TEXT				S(DELTEXT)
EC_KEYWORD				S(DELECKW)
ENTITY_CLASS	N(FNDECH)			S(DELEC)
ENTITY_NAME	N(SELECNM)			S(DELECNM)
INHERITED_ATT_USE				S(DELIAUK)
KEY_CLASS	N(DELMDKC)			S(DELKC)
KEY_CLASS_MEMBER				S(DELKCHT)
MODEL_CLASS	N(VERMOD)			S(DELMOD)
OWNED_ATTRIBUTE				S(DELOACE)
RC_KEYWORD				S(DELRCKW)
RELATION_CLASS	N(DELMDRC)			S(DELRC)

**3.2.8.36 DROP RECORD - Delete the Record Type/Table/Segment from the Internal Schema database.**

**A. Function:**

Drop Record performs the following functions:

1. Deletes all references to the Record Type/Table/Segment from the internal schema portion of the CDM.
2. Deletes all associated Data Fields, Segment data fields, database Areas, project data fields, record keys, record key members, data field linkage, record sets and record set members. Additionally, all text descriptions for the record type/segment data field, and record sets are deleted.

**B. CDM Requirements:**

The Record/Table/Segment to be dropped and the database the Record Type must exist in the CDM database.

**C. Processing:**

1. Drop Record verifies the existence of the database/PCB specified and the record type/table/segment specified. If the database or record type does not exist, processing stops and an error message is issued.
2. The Record/Table/Segment is deleted from the CDM; all associated textual description about the record are deleted; and the record's object number is added to the reusable number table.
3. The process then queries for and deletes all database area assignments associated with the record/table/segment. All data fields that belong to the record are deleted along with their associated textual description. The process then deletes all reference to the record/table/segment in the Project data field, Record key and Record key member tables.

DS 620141100  
1 November 1985

4. Specialized processing for IMS databases deletes the record from IMS SEGMENT\_SIZE and the Segment Data Field CDM tables. For TOTAL databases the DF\_SET\_LINKAGE table has all reference to the record removed.
5. Final processing is to delete all Record\_Sets or Record Set members that contain the record/table/segment to be dropped. The Record Set processing will delete all reference to the Record/table/segment where it is an owner record or member record and any set that becomes memberless when the dropped record was the set member.

DS 620141100  
1 November 1985

CDM TABLES ACCESSED - DROP RECORD

(S=SQL, N=NDML)

<u>TABLE NAME</u>	<u>SELECT</u>	<u>MODIFY</u>	<u>INSERT</u>	<u>DELETE</u>
AUC_ST_MAPPING				S(DELASM2)
DB_AREA_ASSIGNMENT				S(DELDAA2)
DATA_BASE	N(VERDBAS)			
DATA_FIELD	S(DELDL2)			S(PRPDF)
DF_SET_LINKAGE				S(DELDL2)
DESC_TEXT				S(DELTEXT)
IMS_SEGMENT_SIZE				S(DELISS2)
PROJECT_DATA_FIELD	N(PDFREC)			S(DELPDFT) S(DEL1PDF)
RC_BASED_REC_SET				S(DELRBR2)
RECORD_KEY				S(DELRKY2)
RECORD_KEY_MEMBER				S(DELRKM2)
RECORD_SET	N(SELRSET)			S(DELRST2)
RECORD_TYPE	N(VERRT)			S(DELRTY2)
SEGMENT_DATA-FIELD				S(DELSDF2)
SET_TYPE_MEMBER	N(SELSTM)			

3.2.8.37 DROP RELATION - Deletes the relation class and all references to the relation class from the CDM database.

A. Function:

The Drop Relation process performs the following functions for one or more relations:

1. verifies that the relation class exists;
2. verifies that the independent and dependent entities exist;
3. deletes the relation class from the CDM;
4. deletes the complete relation;
5. deletes all keys that have migrated from the relation class;
6. deletes all keywords associated with the relation;
7. deletes all textual descriptions associated with the relation.

B. CDM Requirements:

The Drop Relation process requires the presence of the Relation Class, Independent Entity, and Dependent Entity within the current model.

C. Processing:

1. The Drop Relation process verifies that the independent entity, dependent entity, and the relation exist in the CDM. If any of these do not exist, an error is issued and processing terminates.
2. The process verifies whether the relation is complete and if so returns the key class which allow the process to determine the migration of the associated items.
3. Utilizing the key class the process deletes all migrating key class member based on the relation

DS 620141100  
1 November 1985

class In turn each key class member is deleted from KEY\_CLASS\_MEMBER, ATTRIBUTE\_USE\_CL, and INHERITED\_ATTRIBUTE\_USE based on its association to the relation.

- 4 After all the key class members have been deleted, the process deletes the association in the complete relation, the relation class itself, any keywords associated with the relation, and all textual descriptions of the relation

DS 620141100  
1 November 1985

CON TABLES ACCESSED - DROP RELATION

S(S-SQL, N-NDML)

<u>TABLE NAME</u>	<u>SELECT</u>	<u>MODIFY</u>	<u>INSERT</u>	<u>DELETE</u>
ATTRIBUTE USE CL				S(DELAUCL)
COMPLETE RELATION	N(VERBCC)			S(DELCFRC)
ENTITY CLASS	N(VERENT)			
ENTITY NAME	N(VERENT)			
INHERITED ATT USE	S(DELNIGRC)			S(DELIAUC)
KEY CLASS MEMBER	N(DRPNIGRC)			S(DELCNT)
RC KEYWORD				S(DELCCKV)
RELATION CLASS	N(VERRC)			S(DELRC)

**3.2.8.38 DROP SET - Drop a Record Set/from the Internal Schema**

**A. Function:**

Drop Set allows the NDDL user to drop the set specified from the internal schema portion of the CDM.

**B. CDM Requirements:**

The set to be dropped must have been already defined to the CDM.

**C. Processing:**

1. The user entered database name is used to determine if the database exists in the CDM at this point. The set name is used along with the database number to determine if the set actually exists. If so, all entries for this set are deleted from the following tables:
  - 1.1 RC\_BASED\_REC\_SET, CS to IS relation class to set mappings.
  - 1.2 AUC\_ST\_MAPPING, CS TO IS attribute to set mappings.
  - 1.3 SET\_TYPE\_MEMBER, all member record types for the set.
  - 1.4 RECORD\_SET, the record set occurrence itself.
  - 1.5 For TOTAL DBMS only, the DF\_SET\_LINKAGE occurrence used to indicate the presence of foreign control keys needed for TOTAL link path traversal.
2. The unique set number is then added to the list of re-usable set numbers (object numbers) maintained on the CDM.



DS 620141100  
1 November 1985

CDM TABLES ACCESSED - DROP SET

S(S-SQL,N-NDML)

<u>TABLE NAME</u>	<u>SELECT</u>	<u>MODIFY</u>	<u>INSERT</u>	<u>DELETE</u>
AUC_ST-MAPPING				S(DELASM2)
DATA_BASE	N(VERDBAS)			
DF_SET_LINKAGE				S(DELDSL2)
RC_BASED_REC_SET				S(DELRDR2)
RECORD_SET	N(VERRSET)			S(DELRST2)
SET_TYPE_MEMBER				S(DELSTM2)

**3.2.8.39 DROP VIEW** - Deletes the view/surrogate entity class (SEC) and all data items, project data items, items and related textual descriptions from the CDM database.

**A. Function:**

The Drop View process performs the following functions:

1. verifies the existence of the view;
2. deletes project data items associated with the view;
3. deletes the views SEC\_RC\_COMPONENT entries based on the VIEW/SEC to relation class mapping;
4. deletes all data items and data item textual descriptions;
5. deletes the VIEW/SEC.

**B. CDM Requirements:**

The Drop View process requires that the view exist in the external schema portion of the CDM database.

**C. Processing:**

1. For each view entered the Drop View process verifies that the view to be dropped exists in the CDM database. If the view does not exist, an error message is issued and processing terminates.
2. The process will delete all project data items and the VIEW/SEC to relation class mappings that are based on the view. The process selects all data items belonging to the view, deletes the textual descriptions and then delete all data items.
3. The process deletes the VIEW/SEC and all associated textual descriptions.

DS 620141100  
1 November 1985

CDM TABLES ACCESSED - DROP VIEW

(S=SQL, N=NDML)

<u>TABLE NAME</u>	<u>SELECT</u>	<u>MODIFY</u>	<u>INSERT</u>	<u>DELETE</u>
DATA_ITEM	N(DRFDIV)			S(DELDIV)
PROJECT_DATA_ITEM				S(DELPDI)
SEC	N(VERVIEW)			S(DELSEC)
SEC_RC_COMPONENT				S(DELSECR)

3.2.8.40 HALT - Terminate the current NDDL session

A. Function:

Halt terminates the current NDDL session.

B. Processing:

If any errors were detected during the NDDL session, an ORACLE rollback is performed. If no errors were detected, an ORACLE commit is performed.

DS 620141100  
1 November 1985

CDM TABLES ACCESSED - HALT

NONE

3.2.8.41 MERGE MODEL - Merge two conceptual models.

A. Function:

Merge Model performs the following functions:

1. merge two models into the first-named model or into a newly created third model.
2. generate NDDL commands on a file to populate either the first model or the third model with the attributes, entities, relations, key classes, key class members, aliases, keywords and descriptions from model one and model two.

B. CDM Requirements:

1. Model one must exist in the CDM.
2. Model two must exist in the CDM.
3. If model three is specified, it must not exist in the CDM.

C. Processing:

1. Verify the existence of model one and model two. Note that processing halts if any verification checks fail.
  - 1.1 If model three was not specified, default model three to model one; otherwise, verify that model three does not exist.
  - 1.2 If model three does not exist, copy everything from model one to model three using the COPY MODEL routines.
2. Build the key class list for all the entities in model two.
3. Next, add all the model two top node entities to model three. If the model two entity does not exist in model one, then COPY ENTITY routines are used to add the entity to model three. Otherwise, COMBINE ENTITY routines are used to combine the model two entity with the model one entity with the

result added to model three.

4. Next, add all the model two dependent entities, attributes, relations, keys, and key class members, aliases keywords, and descriptions to model three, level by level. If the model two entity, attribute and/or relation does not exist in model one, then COPY ENTITY routines are used to add the information to model three. Otherwise, COMBINE ENTITY routines are used to combine the model two information with model one with the result added to model three.

DS 620141100  
1 November 1985

CDM TABLES ACCESSED - MERGE MODEL

(S=SQL,N=NDML)

<u>TABLE NAME</u>	<u>SELECT</u>	<u>MODIFY</u>	<u>INSERT</u>	<u>DELETE</u>
ENTITY_CLASS	S(MRGNODE) S(MRGMOD2)			
ENTITY_NAME	N(SELECNM)			
MODEL_CLASS	S(MRGNODE) S(MRGMOD2)			
RELATION_CLASS	S(MRGNODE) S(MRGMOD2)			



3.2.8.42 RENAME - Rename object names for a particular object type.

A. Function:

Rename performs the following functions:

1. change an old object name to a new object name for object types - entity, attribute, keyword, model, domain, view and relation.

B. CDM Requirements:

The object names to be renamed must exist in the CDM.

C. Processing:

1. Rename verifies the existence of the old object name. To rename a relation class, the independent entity, relation name and dependent entities existence is verified.
2. Next, it verifies that the new object name(s) does not previously exist for the particular object type.
3. Finally, the old object name is updated in the CDM with the new object name.
4. Processing halts if any of the verification checks fail.

DS 620141100  
1 November 1985

CDM TABLES ACCESSED - RENAME

(S=SQL, N=NDML)

<u>TABLE NAME</u>	<u>SELECT</u>	<u>MODIFY</u>	<u>INSERT</u>	<u>DELETE</u>
ATTRIBUTE_CLASS	N(VERATT)			
ATTRIBUTE_NAME	N(VERATT)	S(UPDACNM)		
DOMAIN_CLASS	N(VERDOM)	S(UPDTDOM)		
ENTITY_CLASS	N(VERENT)			
ENTITY_NAME	N(VERENT)	S(UPDECNM)		
KEYWORD	N(VERKW)	S(UPDTKW)		
MODEL_CLASS	N(VERMOD)			
MODEL_NAME		S(UPDRCNM)		
RELATION_CLASS	N(VERRC)	S(UPDRCNM)		
SEC	N(VERVIEW)	S(UPDVIEW)		

### 3.3 Performance Requirements

#### 3.3.1 Programming Methods

Structured Design, structured code walkthroughs and structured programming will be used wherever possible. Debugging through use of a symbolic debugger will also be used.

#### 3.3.2 Program Organization

NDDL processor will be organized as a single executable image. It will use the forms processor directly. The DBMS access will be done through ORACLE services communicating with the actual DBMS processes. The NDML will be used for database access and it is designed to communicate via the IISS NTM to the actual request processors. The NDDL processor will consist of a main routine, an initialization routine to establish all the environments, a command initialization routine to clear the parser command processor interface data structure, a command processor entry point for each command and a termination routine.

#### 3.3.3 Modification Consideration

It would be useful to investigate the creation of a separate process for each command processor. Each would have its own processor. UI function screen or menu interface would allow command selection. A forms driven, rather than syntax driven approach could also be considered. Continued evolution of NDML facilities should be monitored, such as generation of "in-line" code, replacement of ORACLE with NDML update facilities and removal of many calls to routines that provide integrity tests, currently coded as separate NDML verification routines, since NDML would generate these. Security considerations for a group of different user types must also be considered. Facilities for displaying the CDM contents must also be considered, specifically generating the NDDL that originally populated the object. The NDDL processor must continually be updated as new features and data tables are added to the CDM.

#### 3.3.4 Special Features

#### 3.3.5 Expandability

The NDDL can be expanded very simply in the area of new commands. Parsing directives must be written and new command

processors designed and implemented without affecting the command processing shell or existing commands

### 3.4 Human Performance

The NDDL processor should allow the CDMA to reliably maintain the most important parts of the CDW and to effectively perform the function of data administration

### 3.5 Database Requirements

#### 3.5.1 Database Overview

The CDW database is relational meaning that it consists of tables that resemble traditional sequential files. The rows of the tables are similar to records in a file and the columns are similar to fields on the records. Columns from different tables are sometimes combined to form 'views' to ease manipulation of the data. The CDW database was derived directly from the definitions found in the CDW-1 model. Reference Number 11

#### 3.5.2 Relations Between Tables and Views

A single complete view has been created for each table of the CDW accessed by NDDL.

#### 3.5.3 Detailed Description of Tables and Views

This following an ORACLE listing of the tables columns of the CDW. By using the table names definitions of each may be found in the CDW 1 mode.

ORACLE DATA DICTIONARY: COL TABLE

TRAME	CHANE	COLTYP	WIDTH	NULLS
AC KEYWORD	AC_NO	NUMBER	0	NOT NULL
	KV_NO	NUMBER	0	NOT NULL
APPLICATION	APPLICATION_MOD_ID	CHAR	10	NOT NULL
	HOST_ID	CHAR	30	NOT NULL
ATTRIBUTE CLASS	AC_NO	NUMBER	0	NOT NULL
	DOMAIN_NO	NUMBER	0	NOT NULL
	MODEL_NO	NUMBER	0	NOT NULL
ATTRIBUTE NAME	AC_NAME	CHAR	30	NOT NULL
	AC_NAME_TYPE	CHAR	8	NOT NULL
	AC_NO	NUMBER	0	NOT NULL
ATTRIBUTE USE CL	AC_NO	NUMBER	0	NOT NULL
	BC_NO	NUMBER	0	NOT NULL
	TAG_NAME	CHAR	30	NOT NULL
	TAG_NO	NUMBER	0	NOT NULL
AUC CONSTRAINT	CONSTRAINT_NO	NUMBER	0	NULL
	STMT_ACTION	CHAR	30	NULL
	TAG_NO	NUMBER	0	NULL
AUC ST MAPPING	AUC_VALUE	CHAR	30	NOT NULL

ORACLE DATA DICTIONARY: COL TABLE

TABLE	CNAME	COLTYP	WIDTH	NULLS
	DB_ID	NUMBER	0	NOT NULL
	SET_ID	CHAR	30	NOT NULL
	TAG_NO	NUMBER	0	NOT NULL
CDM_VERSION	RELEASE_NO	NUMBER	0	NULL
COMPLETE_RELATION	KC_NO	NUMBER	0	NOT NULL
	RC_NO	NUMBER	0	NOT NULL
COPY_LIBRARY	LIBRARY_NAME	CHAR	30	NOT NULL
COPY_MACRO	LIBRARY_NAME	CHAR	30	NOT NULL
MACRO_NAME		CHAR	8	NOT NULL
	MACRO_PURPOSE	CHAR	80	NOT NULL
DATA_BASE	DBMS_NAME	CHAR	30	NOT NULL
	DB_ID	NUMBER	0	NOT NULL
	DB_NAME	CHAR	30	NOT NULL
	HOST_ID	CHAR	30	NOT NULL
DATA_BASE_AREA	AREA_ID	CHAR	30	NOT NULL
	DB_ID	NUMBER	0	NOT NULL
DATA_ELEMENT	DE_NAME	CHAR	30	NULL
	ND	NUMBER	0	NULL
	PIC_SIZE	NUMBER	0	NULL
	PURPOSE	CHAR	240	NULL
	TYPE	CHAR	1	NULL

ORACLE DATA DICTIONARY: COL TABLE

TNAME	CNAME	COLTYP	WIDTH	NULLS
DATA_FIELD	COMPONENT_OF	NUMBER	0	NULL
	DATA_TYPE_NAME	CHAR	30	NULL
	DB_ID	NUMBER	0	NULL
	DF_ID	CHAR	30	NULL
	DF_NO	NUMBER	0	NULL
	OCCURS	NUMBER	0	NULL
	REC_KEY_CODE	CHAR	1	NULL
	REC_SEQ_NO	NUMBER	0	NULL
	REDEF_DF_NO	NUMBER	0	NULL
	RT_ID	CHAR	30	NULL
DATA_ITEM	DATA_TYPE_NAME	CHAR	30	NOT NULL
	DI_ID	CHAR	30	NOT NULL
	DI_NO	NUMBER	0	NULL
DATA_TYPE	VIEW_NO	NUMBER	0	NOT NULL
	TYPE_DESC	CHAR	60	NOT NULL
DBMS	TYPE_ID	CHAR	1	NOT NULL
	DBMS_NAME	CHAR	30	NOT NULL
DBMS_COPY_LIBRARY	DB MODEL	CHAR	1	NOT NULL
	DBMS_NAME	CHAR	30	NOT NULL
DBMS_ON_HOST	LIBRARY_NAME	CHAR	30	NOT NULL
	DBMS_NAME	CHAR	30	NOT NULL
DB_AREA_ASSIGNMENT	HOST_ID	CHAR	30	NOT NULL
	AREA_ID	CHAR	30	NOT NULL
	DB_ID	NUMBER	0	NOT NULL
DB_PASSWORD	RT_ID	CHAR	30	NOT NULL
	DB_ID	NUMBER	0	NOT NULL
	DB_PASSWORD	CHAR	30	NOT NULL
DESCRIPTION_TYPE	DESC_TYPE	CHAR	30	NOT NULL
	DESC_TEXT	CHAR	80	NULL
	DESC_TYPE	CHAR	30	NOT NULL

DS 620141100  
1 November 1985

ORACLE DATA DICTIONARY: COL TABLE

TRNAME	CHNAME	COLTYP	WIDTH	NULLS
	LINE_NO	NUMBER	0	NOT NULL
	OBJECT_NO	NUMBER	0	NOT NULL
	OBJECT_TYPE	CHAR	30	NOT NULL
DF_SET_LINKAGE	DS_ID	NUMBER	0	NOT NULL
	DF_ID	CHAR	30	NOT NULL
	LINKAGE_TYPE	CHAR	1	NOT NULL
	RT_ID	CHAR	30	NOT NULL
	SET_ID	CHAR	30	NOT NULL
DOMAIN_CLASS	DOMAIN_NAME	CHAR	30	NULL
	DOMAIN_NO	NUMBER	0	NULL
DOMAIN_RANGE	BEGIN_VALUE	CHAR	30	NULL
	DOMAIN_NO	NUMBER	0	NOT NULL
	END_VALUE	CHAR	30	NULL



ORACLE DATA DICTIONARY: COL TABLE

TNAME	CNAME	COLTYP	WIDTH	NULLS
DOMAIN_VALUE	DOMAIN_NO	NUMBER	0	NOT NULL
	SPECIFIC_VALUE	CHAR	30	NULL
ECSTUD	DF_NO	NUMBER	0	NULL
	EC_NO	NUMBER	0	NULL
	UNION_VALUE	CHAR	30	NULL
EC_CONSTRAINT	CONSTRAINT_NO	NUMBER	0	NULL
	EC_NO	NUMBER	0	NULL
	STMT_ACTION	CHAR	30	NULL
EC_KEYWORD	EC_NO	NUMBER	0	NOT NULL
	KW_NO	NUMBER	0	NOT NULL
ENTITY_CLASS	EC_NO	NUMBER	0	NOT NULL
	MODEL_NO	NUMBER	0	NOT NULL
ENTITY_NAME	EC_NAME	CHAR	30	NOT NULL
	EC_NAME_TYPE	CHAR	8	NOT NULL
	EC_NO	NUMBER	0	NOT NULL
GENERATED_AP	GENERATED_MOD_ID	CHAR	10	NOT NULL
	GENERATOR_MOD_ID	CHAR	10	NOT NULL
GENERATED_AP_PSB	GENERATED_MOD_ID	CHAR	10	NOT NULL
	PSB_NAME	CHAR	8	NOT NULL

ORACLE DATA DICTIONARY: COL TABLE

TNAME	CNAME	COLTYP	WIDTH	NULLS
HORIZONTAL_PART	CONSTRAINT_NO	NUMBER	0	NULL
	DB_ID	NUMBER	0	NULL
	EC_NO	NUMBER	0	NULL
	RT_NO	NUMBER	0	NULL
HOST	HOST_ID	CHAR	30	NOT NULL
	DB_ID	NUMBER	0	NOT NULL
	RT_ID	CHAR	30	NOT NULL
	SEGMENT_SIZE	NUMBER	0	NOT NULL
INHERITED_ATT_USE	KCM_TAG_NO	NUMBER	0	NOT NULL
	KC_NO	NUMBER	0	NOT NULL
	RC_NO	NUMBER	0	NOT NULL
	TAG_NO	NUMBER	0	NOT NULL
KEYWORD	KEYWORD	CHAR	30	NOT NULL
	KW_NO	NUMBER	0	NOT NULL
KEY_CLASS	EC_NO	NUMBER	0	NOT NULL
	KC_NAME	CHAR	30	NOT NULL
	KC_NO	NUMBER	0	NOT NULL
KEY_CLASS_MEMBER	KC_NO	NUMBER	0	NOT NULL
	TAG_NO	NUMBER	0	NOT NULL
MACRO_CODE	LIBRARY_NAME	CHAR	30	NULL
	MACRO_CODE	CHAR	72	NULL
	MACRO_LINE_NO	NUMBER	0	NULL
	MACRO_NAME	CHAR	8	NULL
MODEL_CLASS	DATE_CREATED	DATE	7	NOT NULL
	DATE_MODIFIED	DATE	7	NOT NULL
	MODEL_NAME	CHAR	30	NOT NULL
	MODEL_NO	NUMBER	0	NOT NULL
	MODEL_STATUS	CHAR	10	NOT NULL

ORACLE DATA DICTIONARY: COL TABLE

TNAME	CNAME	COLTYP	WIDTH	NULLS
NEW_GAP	CASE_NO	NUMBER	0	NULL
	DB_ID	NUMBER	0	NULL
	GENERATED_BY	CHAR	10	NULL
	GENERATED_MOD_ID	CHAR	10	NULL
	GEN_DATE	DATE	7	NULL
	IS_ACTION	CHAR	1	NULL
	MODULE_TYPE	CHAR	10	NULL
	USER_MOD_ID	CHAR	10	NULL
NEXT_NUMBER	AC_NO	NUMBER	0	NOT NULL
	NEXT_NO	NUMBER	0	NOT NULL
	OBJECT_NAME	CHAR	30	NULL
OWNED_ATTRIBUTE	AC_NO	NUMBER	0	NOT NULL
	EC_NO	NUMBER	0	NOT NULL
PROJECT_DATA_FIELD	DATA_TYPE_NAME	CHAR	30	NOT NULL
	DB_ID	NUMBER	0	NOT NULL
	DF_ID	CHAR	30	NOT NULL
	PRIM_SECONDARY	CHAR	1	NOT NULL
	RT_ID	CHAR	30	NOT NULL
	TAG_NO	NUMBER	0	NOT NULL
PROJECT_DATA_ITEM	DI_ID	CHAR	30	NOT NULL
	PRIM_SECONDARY	CHAR	1	NOT NULL
	TAG_NO	NUMBER	0	NOT NULL
	VIEW_NO	NUMBER	0	NOT NULL
	HOST_ID	CHAR	30	NOT NULL
PSB	PSB_NAME	CHAR	8	NOT NULL
PSB_PCB	DB_ID	NUMBER	0	NOT NULL
	KEY_FEEDBACK_LEN	NUMBER	0	NOT NULL
	PCB_SEQ_NO	NUMBER	0	NOT NULL
	PSB_NAME	CHAR	8	NOT NULL

ORACLE DATA DICTIONARY: COL TABLE

TNAME	CNAME	COLTYP	WIDTH	NULLS
RC_BASED_REC_SET	DB_ID	NUMBER	0	NOT NULL
	RC_NO	NUMBER	0	NOT NULL
	RT_ID	CHAR	30	NOT NULL
	SET_ID	CHAR	30	NOT NULL
RC_KEYWORD	KW_NO	NUMBER	0	NOT NULL
	RC_NO	NUMBER	0	NOT NULL
RECORD_KEY	DB_ID	NUMBER	0	NOT NULL
	REC_KEY_DEC_LEN	NUMBER	0	NULL
	REC_KEY_ID	NUMBER	0	NOT NULL
	REC_KEY_LABEL	CHAR	30	NOT NULL
	REC_KEY_UNI_IND	NUMBER	0	NOT NULL
	REC_KEY_VALUE_LEN	NUMBER	0	NULL
	RT_ID	CHAR	30	NOT NULL
RECORD_KEY_MEMBER	TYPE_ID	CHAR	1	NULL
	DB_ID	NUMBER	0	NOT NULL
	DF_ID	CHAR	30	NOT NULL
	REC_KEY_ID	NUMBER	0	NOT NULL
	RT_ID	CHAR	30	NOT NULL
RECORD_SET	SEQ_NO	NUMBER	0	NOT NULL
	DB_ID	NUMBER	0	NOT NULL
	RT_ID OF_OWNER	CHAR	30	NOT NULL
	SET_ID	CHAR	30	NOT NULL
	SET_NO	NUMBER	0	NULL
RECORD_TYPE	TOTAL_NUM_MEM	NUMBER	0	NOT NULL
	DB_ID	NUMBER	0	NOT NULL
	RT_ID	CHAR	30	NOT NULL
RECORD_TYPE_COMP	RT_NO	NUMBER	0	NULL
	DB_ID	NUMBER	0	NOT NULL
RELATION_CLASS	EC_NO	NUMBER	0	NOT NULL
	RT_ID	CHAR	30	NOT NULL
	DEP_EC_NO	NUMBER	0	NOT NULL
	IND_EC_NO	NUMBER	0	NOT NULL
	MAX_NO_DEP_ENT	NUMBER	0	NOT NULL
	MIN_NO_DEP_ENT	NUMBER	0	NOT NULL
	NO_IND_ENT	NUMBER	0	NOT NULL

ORACLE DATA DICTIONARY: COL TABLE

TNAME	CNAME	COLTYP	WIDTH	NULLS
	RC_NAME	CHAR	30	NOT NULL
	RC_NO	NUMBER	0	NOT NULL
REUSABLE_NUMBER	AC_NO	NUMBER	0	NOT NULL
	REUSE_NO	NUMBER	0	NOT NULL
SCHEMA_NAMES	DB_ID	NUMBER	0	NOT NULL
	DB_LOCATION	CHAR	30	NULL
	SCHEMA_NAME	CHAR	30	NOT NULL
	SUBSCHEMA_NAME	CHAR	30	NOT NULL
SEC	SEC_ID	CHAR	30	NOT NULL
	VIEW_NO	NUMBER	0	NOT NULL
SEC_RC_COMPONENT	RC_NO	NUMBER	0	NOT NULL
	VIEW_NO	NUMBER	0	NOT NULL
SEGMENT_DATA_FIELD	DB_ID	NUMBER	0	NOT NULL
	DF_ID	CHAR	30	NOT NULL
	INS_DF_IND	CHAR	1	NOT NULL
	RT_ID	CHAR	30	NOT NULL
	SEG_START_BYTE	NUMBER	0	NOT NULL
SET_TYPE_MEMBER	DB_ID	NUMBER	0	NOT NULL
	REQ_MEM_IND	CHAR	1	NOT NULL
	RT_ID_OF_MEMBER	CHAR	30	NOT NULL
	SET_ID	CHAR	30	NOT NULL
SOFTWARE_MODULE	LANG_NAME	CHAR	10	NOT NULL
	LATEST_REV_DATE	NUMBER	0	NOT NULL
	LATEST_USAGE_DATE	NUMBER	0	NOT NULL
	MOD_ABSTRACT	CHAR	60	NOT NULL
	MOD_ID	CHAR	10	NOT NULL
	MOD_TITLE	CHAR	30	NOT NULL
	STATUS_IND	CHAR	1	NOT NULL
SOFTWARE_SEC	MOD_ID	CHAR	10	NOT NULL
	SEC_ID	CHAR	30	NOT NULL
SOFTWARE_SUB	CALLING_MOD_ID	CHAR	10	NOT NULL

DS 620141100  
1 November 1985

ORACLE DATA DICTIONARY: COL TABLE

TNAME	CNAME	COLTYP	WIDTH	NULLS
	SUBROUTINE MOD_ID	CHAR	10	NOT NULL
USER_DEF_DATA_TYPE	DATA_TYPE_IND	CHAR	4	NULL
	DATA_TYPE_NAME	CHAR	30	NULL
	DOMAIN_NO	NUMBER	0	NULL
	MAX SIZE	NUMBER	0	NULL
	NO OF DECIMALS	NUMBER	0	NULL
	TYPE_ID	CHAR	1	NULL
	USDF_DT_NO	NUMBER	0	NULL
VERIF_MODULE	DOMAIN_NO	NUMBER	0	NULL
	MOD_ID	CHAR	10	NULL

## SECTION 4

### QUALITY ASSURANCE PROVISIONS

#### 4.1 Introduction and Definitions

"Testing" is a systematic process that may be preplanned and explicitly stated. Test techniques and procedures may be defined in advance and a sequence of test steps may be specified. "Debugging" is the process of isolation and correction of the cause of and error.

#### 4.2 Computer Programming Test and Evaluation

The quality assurance provisions for test will consist of the normal testing techniques that are accomplished during the construction process. They consist of design and code walk-throughs, unit testing, and integration testing. These tests will be performed by the design team.

The integration test developed for the NDDL will consist of a list of commands (and their expected outputs) which will be used by the tester. This session will test each command to ensure its correct operation. Results of the session may be compared with those of the unit testing.

Because rather flat hierarchy of modules is designed for the NDDL, unit testing will primarily involve testing each of the NDDL interface routines and internal functions for correct processing and output. Below the level of modules implementing each command will be a small set of procedures for database commit and rollback and error handling.

DS 620141100  
1 November 1985

SECTION 5

PREPARATION FOR DELIVERY

The implementation site for the constructed software will be the ICAM Integrated Support System (IISS) Test Bed Site located in Albany, New York. The software associated with the WDDL will be clearly identified and will include instructions on procedures to be followed for installation of the release.



END

8-87

DTIC