LOAN DOCUMENT

PHOTOGRAPH THIS SHEET

WRDC-TR-90-5024

DOCUMENT IDENTIFICATION:

NOV 1990

DISTRIBUTION STATEMENT A

Approved for public release;
Distribution Unlimited

DISTRIBUTION STATEMENT

DTIC ELECTE

OCT 25 1990

DATE ACCESSIONED

DATE RETURNED

90 1 130

DATE RECEIVED IN DTIC

REGISTERED OR CERTIFIED NUMBER

PHOTOGRAPH THIS SHEET AND RETURN TO DTIC-FDAC

DTIC FILE COPY

LEVEL

INVENTORY

HANDLE WITH CARE

DISTRIBUTION CODES

1

TRBM

NAV

AMM

SPAA

DATE ACCESSIONED

A - 1

DISTRIBUTION STAMP

90 1 130

DATE RECEIVED IN DTIC

REGISTERED OR CERTIFIED NUMBER
NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely Government-related procurement, the United States Government incurs no responsibility or any obligation whatsoever. The fact that the government may have formulated or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication, or otherwise in any manner construed, as licensing the holder, or any other person or corporation; or as conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

This report 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.

Charles Gilman
Project Manager

Walter H. Reimann, Chief
Computer-Integrated Mfg. Branch

FOR THE COMMANDER

BRUCE A. RASMUSSEN
Chief, Integration Technology Division
Manufacturing Technology Directorate

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 WRC/MT, WPAFB, OH 45433-6533 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.
**Geometric Modeling Applications Interface Program**

**This User's Manual provides a guide for the use of Model Access Software for the Computer Program Configuration Item (CPCI) identified as the GMAP (Geometric Modeling Applications Interface Program). U.S. Air Force Contract F33615-85-C-5122. It includes descriptions of the Model Access Software and Name Value Interface capabilities.**

**ABSTRACT**

18. Subject Terms (Continued)
   Product Life Cycle
   Engineering
   Manufacturing
   Interface
   Exchange Format
   CAD
   CAM
   CIM
   IBIS
   RFC
   System Translator
   Schema Manager
   Model Access Software
   Name/Value Interface
FOREWORD

This Model Access Software User's Manual describes work performed under Air Force Contract F33615-85-C-5122, Geometric Modeling Applications Interface Program (GMAP), covering the period 1 August 1985 to 31 July 1988. This User's Manual provides a guide for the use of Model Access Software under this contract which is sponsored by the Computer Integrated Manufacturing Branch, Materials Laboratory, Air Force Systems Command, Wright Air Force Base, Ohio 45433-6533. The GMAP Project Manager for the Air Force is Mr. Charles Gilman.

The primary contractor is Pratt & Whitney, an operating unit of United Technologies Corporation. Mr. Richard Lopatka is managing the GMAP project at Pratt & Whitney. Ms. Linda Phillips is the Program Integrator. Mr. John Hamill is the Deputy Program Manager.

McDonnell Aircraft Company is the subcontractor responsible for the Model Access Software work. Mr. Jerry Weiss is the GMAP Program Manager at McDonnell Aircraft and Mr. Herb Ryan is the Deputy Program Manager.

NOTE: The number and date in the upper right corner of each page in this document indicate that it has been prepared in accordance to the ICAM CH Life Cycle Documentation requirements for a Configuration Item (CI).
TABLE OF CONTENTS

SECTION 1. SCOPE

| 1.1 Identification | 1-1 |
| 1.2 Introduction | 1-1 |
| 1.3 Other System Manuals | 1-1 |
| 1.4 Approach | 1-2 |

SECTION 2. REFERENCES

| 2.1 Reference Documents | 2-1 |
| 2.1.1 Military | 2-1 |
| 2.1.2 Commercial | 2-3 |
| 2.1.3 Standards Organizations | 2-4 |
| 2.2 Terms and Acronyms | 2-4 |
| 2.2.1 Terms Used In GMAP | 2-4 |
| 2.2.2 Acronyms Used In GMAP | 2-18 |

SECTION 3. SYSTEM OVERVIEW

| 3.1 System Architecture | 3-1 |
| 3.1.1 System Interfaces | 3-1 |
| 3.1.2 System Environment | 3-1 |
| 3.2 Schema Manager | 3-3 |
| 3.2.1 Physical Schema | 3-3 |
| 3.3 EF | 3-3 |
| 3.4 System Translator | 3-3 |
| 3.5 Model Access Software | 3-3 |
| 3.5.1 Data Item | 3-4 |
| 3.5.1.1 Entity | 3-4 |
| 3.5.1.2 List | 3-6 |
| 3.5.2 Interface Parameters | 3-6 |
| 3.5.2.1 Data-Name Parameters | 3-6 |
| 3.5.2.2 Data-Type Parameters | 3-7 |
| 3.5.2.3 Formal Data Types | 3-7 |
| 3.5.3 Memory Manager | 3-8 |
| 3.6 NVI | 3-8 |

SECTION 4. MODEL ACCESS SOFTWARE (MAS) OPERATIONS AND ENVIRONMENT

| 4.1 Introduction | 4-1 |
| 4.2 Initialization/Deletion of the MAS Working Form (WF) | 4-2 |
| 4.3 Entity Operations | 4-5 |
| 4.3.1 Create Operations | 4-5 |
| 4.3.2 Query Operations | 4-8 |
| 4.3.3 Update Operations | 4-12 |
| 4.3.4 Delete Operations | 4-14 |
**TABLE OF CONTENTS (contd.)**

| 4.3.5  | Activate Operations                                      | 4-21 |
| 4.3.6  | Application Flag Operations                              | 4-25 |
| 4.4    | List Operations                                          | 4-38 |
| 4.4.1  | Create Operations - Application Lists                    | 4-39 |
| 4.4.2  | Query Operations - Application Lists and Constituent Lists | 4-54 |
| 4.4.3  | Update Operations - Application Lists and Constituent Lists | 4-61 |
| 4.4.4  | Update Operations - Application Lists Only               | 4-68 |
| 4.4.5  | Boolean Operations - Application Lists and Constituent Lists | 4-73 |
| 4.4.6  | Delete Operations - Application Lists Only               | 4-77 |
| 4.5    | Execute Operations                                       | 4-82 |
| 4.6    | General Purpose Utilities                                | 4-94 |
| 4.7    | Special Purpose Utilities                                | 4-99 |
| 4.8    | IBM/MVS Environment                                      | 4-105 |
| 4.8.1  | Compiling Considerations                                 | 4-105 |
| 4.8.2  | Considerations When Using the XEQ Routines (MAEXEQ, MALXEQ, MAEXE, MAEXQ, MAEXUQ, MALSRT) | 4-105 |
| 4.8.3  | Linking Considerations                                   | 4-106 |
| 4.9    | VAX/VMS Environment                                      | 4-106 |
| 4.9.1  | Compiling Considerations                                 | 4-106 |
| 4.9.2  | Considerations When Using the XEQ Routines (MAEXEQ, MALXEQ, MAEXE, MAEXQ, MAEXUQ, MALSRT) | 4-107 |
| 4.9.3  | Linking Considerations                                   | 4-108 |

**SECTION 5.**

| 5.1    | Overview                                               | 5-1  |
| 5.2    | Direct Query/Store                                      | 5-1  |
| 5.2.1  | Function                                               | 5-1  |
| 5.2.2  | Direct Query Format                                    | 5-3  |
| 5.2.3  | Direct Store Format                                     | 5-4  |
| 5.3    | Procedural Query                                        | 5-4  |
| 5.3.1  | Function                                               | 5-4  |
| 5.3.2  | Format                                                 | 5-6  |
| 5.4    | Utilities                                              | 5-7  |
| 5.4.1  | Function                                               | 5-7  |
| 5.4.2  | Attribute Data Type Query Format                        | 5-7  |
| 5.5    | IBM/MVS Environment                                    | 5-9  |
| 5.5.1  | Compiling Considerations                               | 5-9  |
| 5.5.2  | Include Files                                           | 5-9  |
## LIST OF ILLUSTRATIONS

<table>
<thead>
<tr>
<th>Figure</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-1</td>
<td>CMAP/PDDI System Architecture</td>
<td>3-2</td>
</tr>
<tr>
<td>3-2</td>
<td>LINE: An Entity With Constituents</td>
<td>3-5</td>
</tr>
<tr>
<td>4-1</td>
<td>MAS Interface Operations</td>
<td>4-1</td>
</tr>
<tr>
<td>4-2</td>
<td>Execute Operation</td>
<td>4-83</td>
</tr>
</tbody>
</table>
## LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4-1</td>
<td>CREATE ROUTINES</td>
<td>4-5</td>
</tr>
<tr>
<td>4-2</td>
<td>QUERY ROUTINES</td>
<td>4-8</td>
</tr>
<tr>
<td>4-3</td>
<td>UPDATE OPERATIONS</td>
<td>4-12</td>
</tr>
<tr>
<td>4-4</td>
<td>DELETE RULES</td>
<td>4-15</td>
</tr>
<tr>
<td>4-5</td>
<td>DELETE ROUTINES</td>
<td>4-16</td>
</tr>
<tr>
<td>4-6</td>
<td>ACTIVATE ROUTINES</td>
<td>4-21</td>
</tr>
<tr>
<td>4-7</td>
<td>APPLICATION FLAG OPERATIONS</td>
<td>4-25</td>
</tr>
<tr>
<td>4-8</td>
<td>CREATE ROUTINES</td>
<td>4-39</td>
</tr>
<tr>
<td>4-9</td>
<td>QUERY OPERATIONS - APPLICATION AND CONSTITUENT LISTS</td>
<td>4-54</td>
</tr>
<tr>
<td>4-10</td>
<td>UPDATE OPERATIONS - APPLICATION AND CONSTITUENT LISTS</td>
<td>4-61</td>
</tr>
<tr>
<td>4-11</td>
<td>UPDATE OPERATIONS - APPLICATION LISTS</td>
<td>4-68</td>
</tr>
<tr>
<td>4-12</td>
<td>BOOLEAN ROUTINES</td>
<td>4-73</td>
</tr>
<tr>
<td>4-13</td>
<td>DELETE ROUTINES</td>
<td>4-77</td>
</tr>
<tr>
<td>4-14</td>
<td>EXECUTE ROUTINES</td>
<td>4-84</td>
</tr>
<tr>
<td>4-15</td>
<td>GENERAL PURPOSE UTILITIES</td>
<td>4-94</td>
</tr>
<tr>
<td>4-16</td>
<td>SPECIAL PURPOSE UTILITIES</td>
<td>4-99</td>
</tr>
</tbody>
</table>
SECTION 1

SCOPE

1.1 Identification

This User's Manual provides a guide for the use of Model Access Software developed for the Product Definition Data Interface (Project 5601) and the Name Value Interface software developed for the Geometric Modeling Applications Interface Program. The Product Definition Data Interface project was developed under Air Force Contract F33516-82-C-5036 and the Geometric Modeling Applications Interface Program was developed under Air Force Contract F33615-85-C-5122.

1.2 Introduction

Model Access Software capabilities documented in this manual include:

- Access Software Initialization
- Entity Operations
- List Operations.

NVI capabilities documented in this manual include:

- Direct Query/Store
- Procedural Query.

This software was developed on IBM 43xx and 308xx computers and migrated to DEC VAX 11/780 and other computers. The environmental requirements are described in Section 3.

This manual does not address local (native) system or computing environment documentation.

This manual addresses IBM procedures and terminology only.

1.3 Other System Manuals

An associated Operator's Manual (OM560240001U) describes the system operation and installation procedures. Procedures are also included for
migrating the software from IBM/MVS to other computer systems (i.e., VAX). The Operator's Manual is intended for use by computer operators and programming personnel.

An associated Translator User's Manual (UM560240021U) is provided for users of the System Translator, and a Schema Manager Users Manual (UM560240011U) is provided for users of the Schema Manager.

The Software Components Product Specification (PS560240032U) provides routine descriptions, data dictionary listings and system messages for system maintenance purposes.

1.4 Approach

This User’s Manual is divided into five main sections and six appendices:

Section 1 - Scope of this document.

Section 2 - Reference documentation applicable to GMAP and this document.

Section 3 - The PDDI/GMAP architecture at a high level and introduction to the use of the Model Access Software.

Section 4 - Entity and List Operations needed to access the data structures passed back to the Application program.

Section 5 - Description of the use of the Name Value Interface.

Appendices:

Appendix A - Model Access Software Calling Parameter Index
Appendix B - Alphabetical Model Access Software Routine Index
Appendix C - Model Access Software Return Code Index
Appendix D - General Techniques/Guidelines
Appendix E - Run Time Environment
Appendix F - Sample Programs.
SECTION 2
REFERENCES

2.1 Reference Documents

The following technical reports, specifications, standards, and other documents have been referred to or are relevant to this Model Access Software User's Manual.

2.1.1 Military:


Interim Technical Report No. 4 (ITR560240004U)
"Geometric Modeling Applications Interface Program" November 1986

Geometric Modeling Applications Interface Program (GMAP) Needs Analysis

Interim Technical Report No. 5 (ITR560240005U)
"Geometric Modeling Applications Interface Program" January 1987
(Period 1 August 1986 - 31 October 1986).

Geometric Modeling Applications Interface Program (GMAP) System

Geometric Modeling Applications Interface Program (GMAP) State of the

Interim Technical Report No. 6 (ITR560240006U)
"Geometric Modeling Applications Interface Program" May 1987
(Period 1 November 1986 - 31 January 1987)

Geometric Modeling Applications Interface Program (GMAP) System
Specification (Volumes I-IV), CI SS560240001U, July 1987

Interim Technical Report No. 7 (ITR560240007U)
"Geometric Modeling Applications Interface Program," August 1987
(Period 1 February 1987 - 30 April 1987).

Geometric Modeling Applications Interface Program (GMAP) System Design

Geometric Modeling Applications Interface Program (GMAP) to Retirement
for Cause Interface Development Specification, CI DS560240011U, November
1987.

Geometric Modeling Applications Interface Program (GMAP) to Integrated
Blade Inspection System Interface Development Specification, CI

Geometric Modeling Applications Interface Program (GMAP) to Retirement
for Cause Interface As-designed Product Specification, CI PS560240011U,
December 1987.

Geometric Modeling Applications Interface Program (GMAP) to Retirement
for Cause Interface Unit Test Plan, CI UTP560240011U, December 1987.
Interim Technical Report No. 8 (ITR560240008U)  
"Geometric Modeling Applications Interface Program," December 1987  

Interim Technical Report No. 9 (ITR560240009U)  
"Geometric Modeling Applications Interface Program," March 1988  
(Period 1 August 1987 – 31 October 1987).


Interim Technical Report No. 10 (ITR560240010U)  
"Geometric Modeling Applications Interface Program," August 1988  

Interim Technical Report No. 11 (ITR560240010U)  
"Geometric Modeling Applications Interface Program," August 1988  


Interim Technical Report No. 12 (ITR560240012U)  
"Geometric Modeling Applications Interface Program" October 1988  

Geometric Modeling Applications Interface Program (GMAP) to Retirement for Cause Interface Unit Test Report, CI UTR560240011U, November 1988.

Geometric Modeling Applications Interface Program (GMAP) System

Geometric Modeling Applications Interface Program (GMAP) to Retirement

Geometric Modeling Applications Interface Program (GMAP) to Integrated
Blade Inspection System Interface As-built Product Specification, CI

Geometric Modeling Applications Interface Program (GMAP) System

Geometric Modeling Applications Interface Program (GMAP) to Integrated
Blade Inspection System Interface User/Operator Manual, CI

Geometric Modeling Applications Interface Program (GMAP) Schema Manager

Interim Technical Report No. 13 (ITR560240013U)
"Geometric Modeling Applications Interface Program" February 1989
(Period 1 August 1988 - 31 October 1988).

Interim Technical Report No. 14 (ITR560240014U)
"Geometric Modeling Applications Interface Program" July 1989

Geometric Modeling Applications Interface Program (GMAP) Model Access

Geometric Modeling Applications Interface Program (GMAP) PDD Editor

Demonstration Model Descriptions for Geometric Modeling Applications
Interface Program (GMAP), CI TTD560240001U, July 1989.

Product Information Exchange System (PIES) User Manual for Geometric
Modeling Applications Interface Program (GMAP), CI TTD560240002U, July
1989.

2.1.2 Commercial

_ A Practical Guide to Splines_, C. de Boor, Applied Mathematical Sciences,
Vol. 27, Springer-Verlag.


2.1.3 Standards Organizations

ANSI Y14.5M, Dimensioning and Tolerancing.


2-5


2.2 Terms and Acronyms

A glossary of terms frequently used in GMAP which may be included in this Model Access Software User's Manual is provided below. Some reference notes applicable to these definitions are presented after the glossary. A list of acronyms and abbreviations used in GMAP is also included in this section.

2.2.1 Terms Used in GMAP

Accept/Reject/Incomplete Notice -- A display on the cell computer that indicates the final status of the engine disk.

Accept = Acceptable within tolerance specified by engine manufacturer
Reject = Rejected because of flaw(s) outside the range of acceptable tolerances
Incomplete = Part cannot be inspected

Access Software -- A set of routines for creating, managing and querying an incore Working Form model.

Angular -- An angular size tolerance is used to tolerance the size of an angular feature independent of its angular location along an arc.

Application -- A method of producing a specific result.

Application Request -- A request initiated by an application program, either through batch or interactive processing, which will interrogate the model through the PDDI Access Software to obtain or operate on specific information regarding the model and its components or elements.

Application Requested Data -- The data which fulfills the application's original request and which is in the proper format and readable by the application.

Architecture -- A design or orderly arrangement.

ASCII -- American Standard Code for Information Interchange.
As-Is -- The present condition.

Attribute -- A quality of characteristics element of any entity having a name and a value.

B-Spline -- A spline defined by a control polygon, B-spline basis functions, and an associated knot vector. A Bezier curve is a special case of a B-spline; a nurbs is the most general case of a B-spline.

Bezier Curve -- A type of curve defined by a set of vertices called a control polygon and a set of basis functions. The basis functions are known as Bernstein polynomials. K vertices define a curve of order K-1.

Blend -- A smooth, continuous transition from one surface to another.

Boundary Representation -- A topology imposed on 3-D geometric entities to yield a general solid model. That model describes an object by describing its boundary area.

Body of Revolution (BOR) Representation -- A topology in which an object is represented as the volume swept by a curve rotated about a line. This is a boundary representation in which the curve represents the surface area of the object.

Bounded Geometry -- Geometry that has limits defined by its mathematical domain or range.

Calibration Block Parameters (Scale Factors) -- Nondestructive test parameters used to adjust a specific cell. These parameters are obtained from the calibration blocks located at each cell.

Circumferential -- A circumferential tolerance specifies the tolerance zone within which the average diameter of a circular feature must lie. The average diameter is the actual circumference divided by pi (3.14159). A circumferential tolerance is a specific example of a peripheral or perimeter tolerance for a general curve.

Class -- A collection of entities that are alike in some manner.

CLIST -- IBM Command lists.

Composite Curve -- A group of curve segments that are C0 continuous.
Compound Feature Representation — An enumerative feature representation in which at least one component is itself a feature. For example, a bolt hole circle might be represented as a list of individual hole features.

Concentricity (Generic) — A concentricity tolerance specifies a cylindrical tolerance zone within which the axis of a feature must lie, where the axis of the zone coincides with the axis of the datum.

Conceptual Schema — Formally specified global view that is processing independent, covering information requirements and formulation of independent information structures. A neutral view of data, usually represented in terms of entities and relations.

Conic — A quadratic curve represented in the most general case by the equation:

\[ Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0. \]

A conic may be a circle, line, ellipse, parabola, or a hyperbola depending on the coefficients, A, B, C, D, E, and F.

Constraints (Generic) — An assertion to explicitly specify data meaning or semantics.1 (Notes appear at the end of this section.)

Context-Free Grammar — The syntax of the language gives a precise specification of the data without interpretation of it.

Constituent — A specific instance of an entity that is used in the definition of some other entity.

Data Dictionary — A catalog of all data elements in a design, giving their name, definition, format, source, and usage. May also include data types and value limits.

Defining Airfoil Sections — A planar or conical section that depicts an airfoil profile. Defining airfoil sections are those that meet aerodynamic requirements. Other intermediate sections are added for Manufacturing purposes.

Dimension — A part dimension is a quantifiable value expressing size, form, or location.

Domain — The set of values permissible in a given context.

Dynamic Allocation — The allocation (and de-allocation) of memory resources as required by the application. The opposite is static allocation where a fixed size segment of memory is available to the application.
Eddy Current Cell — Hardware used to perform an Eddy current inspection operation (surface flaws).

Eddy Current Inspection — An inspection method used to detect internal potential flaws on a disk. It is based on the principle of sending electromagnetic signals to a target area on a part and detecting/interpreting reflection (Eddy current) from the target.

Eddy Current Scan Plan — An interpreter code program controlling the Eddy current inspection of a particular geometry.

Eddy Current/Ultrasonic Flaw Data Printout — A printout containing size and location information about specific flaw(s) (both critical and noncritical) associated with a particular part.

Entity — A description of a person, place, or thing, about which information is kept.

External Reference — A reference to some quantity of data that exists somewhere outside the scope of the immediate body of information.

Feature — A part feature in the dimensioning and tolerancing context is a feature in the sense of ANSI Y14.5M, that is, a physical component portion of a part, such as a surface, hole, slot, and so on, that is used in a tolerancing situation. In the dimensioning and tolerancing context, a feature consists of individual or groups of basic shape elements used to define the physical shape of an item. This general dimensioning and tolerancing use of features is to be distinguished from Features. The word "features" alone implies dimensioning and tolerancing features. The term "form feature" is described below.

Feature Pattern — A geometric pattern of occurrences of similar form features, for example, a circular pattern of scallops, a rectangular array of holes.

Feature Representation (Generic) — A description of a form feature within the context of a geometric model.

Feature Type — A name applied to a form feature that is suggestive of its shape and size, for example, hole, slot, web.

Feature of Size (Generic) — A feature of size provides a geometric location capable of being referenced for use with datums and tolerances. A feature of size can be a GMAP feature, or other referenceable shape elements of a part model that are symmetric about a point, line, plane, axis, curve, and so on. When a feature of size is used in a relationship with a tolerance or datum, its feature of symmetry is the implied reference.
Flat Pattern Representation (Extrusion Representation) -- A topology in which an object is represented as the volume swept by a planar polygon moving in a direction normal to its plane. The polygon may have internal polygon represent the surface area of the object.

Flaw Characteristics -- Location, length, width, depth, and nondestructive test parameters associated with a specific flaw.

Flaw Data Packet -- Packet containing nonevaluated flaw data. Note that the packet can contain zero flaws.

Flaw Orientation -- The direction of the major characteristic of the flaw with respect of the part coordinate system. (See the notes section at the end of this glossary.)

Flaw Suspect Location -- The coordinate location of a possible flaw detected during a survey mode inspection (six-axis position of ultrasonic cell, seven-axis position of Eddy current cell).

Form Feature -- A portion of a part's geometry that is useful to regard as an entity. In a boundary representation context, this is a subset of the part's surface area.

Form Tolerance -- Form tolerances are used to control the form of model features. A form tolerance specifies the amount that an actual features form may vary from nominal. Form tolerance include straightness tolerance, flatness tolerance, roundness/circularity tolerance, cylindricity tolerance, perpendicularity tolerance, parallelism tolerance, angularity tolerance, profile-of-a-line tolerance, profile-of-a-surface tolerance, circular-runout tolerance, true-direction tolerance, and mismatch tolerance.

Functionality -- (1) To show that the configuration item has fulfilled the specified requirements. (2) The receiving and sending systems can operate on the entity in the same manner with the same results within a pre-defined tolerance.

Function Modeling -- A description of a system in terms of a hierarchy of functions or activities, each level decomposing higher ones into greater detail. Functions are named by verbs; nouns related are declared as inputs, controls, outputs, and mechanisms.

Geometric Element (Generic) -- An instance of a geometric entity.

Geometric Group -- A group of geometric entities with a name.

Geometric Model -- A part description in terms of its underlying geometric elements. The model may be a wireframe, surface, or solid model.
Geometric Pattern -- A circular or rectangular pattern of geometric entities.

Group Technology Code -- An alphanumeric string identifying significant characteristics of a product, enabling group technology applications. Also known as Part Classification Code.

Include File -- PASCAL source code from another file or library included on the compilation of a PASCAL source file.

Input Data -- That information which the application needs to supply in order to interrogate or operate on the model. This data may assume only these forms prescribed by the PDDI Access Software specification.

Inspection Cycle -- A period for which nondestructive testing inspection requirements are defined.

Inspection Cycle Zone -- An entity that is composed of a unique combination of zone and inspection cycle.

Inspection Module Operator -- Refers to personnel operating RFC cell(s).

Instrument Setting Adjustments -- Nondestructive testing parameter adjustments automatically accomplished via pre- and post-calibration operations. These adjustments have to be accomplished within a predetermined tolerance.

Internal Flaw -- A subsurface anomaly.

Internal Flaw Major Characteristic -- A vector determined by an agreed upon method.

Example (1): The vector of greatest magnitude from the centroid to a boundary of the anomaly.

Example (2): A vector representing the major axis of the minimum ellipsoidal envelope encompassing the anomaly.

Internal Flaw Tolerance -- A unique combination of:

(a) Internal flaw orientation range.
(b) Serviceable internal flaw tolerance limits.
(c) Repairable internal flaw tolerance limits.
Internal Flaw Tolerance Limit -- A unique combination of:

7(a) Maximum diameter.
(b) Maximum depth below surface.
(c) Maximum thickness.

Interpreted Request -- Input data which has been appropriately modified to conform to the PDDI Access Software's internal data representation so that it may be further processed.

Key Attribute -- An attribute or combination of attributes having values that uniquely identify each entity instance.\(^2\)

Laminates Representation (Generic) -- A topology in which an object is represented as layers of flat material of known thickness.

Location Tolerance -- Location tolerances specify the allowable variation in position of model features. Location tolerances include various forms of position tolerancing conventions. These are (true) position, concentricity, alignment, rectilinear location, and angular location.

Logistics Support -- The function of procuring, distributing, maintaining, replacing, and repairing material in support of a delivered product.

Machine Coordinate Positions -- The probe location with respect to machine coordinates.

Machine Preset Data -- Machine coordinate adjustments automatically accomplished via pre- and post-calibration operations. These adjustments have to be accomplished within predetermined tolerance.

Metadata -- Data about data. Defines the physical schema and record formats of the part data.

Metamodel -- A body of data that defines the characteristics of a data model or structure.

Model -- A collection of PDD that is transferable, displayable, accessible, and equivalent to a Part. The internal representation of the application data, as initiated and organized by the user. The model is also referred to as the Working Form.

Model Network Definition -- The set of rules and definitions which outline in detail the data structure whereby higher order entities may be composed of lower order entities, or constituents, and the lower order entities may be constituents of one or more higher order entities.
Native System -- The PDD and applications in a format that is unique to the database of a CAD system.

Nondestructive Testing Parameters -- Parameters used by the Eddy current and ultrasonic instruments (examples: amplitude, phase angle, gain, threshold, and so on).

Nonconstructive Feature Representation (Explicit Feature Representation) -- A feature representation that at least partially depends on a declaration that a face, or portion of a face, it "in" the feature.

Nondestructive Testing Personnel -- Personnel responsible for the generation of scan plans and derivation of applicable nondestructive testing instrument settings used in the scan plans.

Nonshape Data -- Produce definition data that cannot be represented by shape elements.

Normal Forms -- Conditions reflecting the degree of refinement and control over the relationships and entities in an information model.

Numerical Control Program (Complete and Proposed) -- Set of program instructions used to generate a probe path.

Orientation Range -- An envelope in which the major flaw characteristic must lie.

Parse -- The process of analyzing input strings (records) to identify fields and to verify that the data has a valid format.

Part Blueprint -- A blueprint provided by the engine manufacturer of a particular F100 engine disk.

Physical Schema -- Internal representation of data; the computer view that includes stored record format and physical ordering of stored records.

PID File -- A PID File is a copy of the Working Form filed to disk for temporary storage. The software that produces this capability (PID Code) is provided as an interim solution while a translator to the native database is in development.

Polynomial Spline -- A parametric spline of order 1, 2, or 3 defined by a set of N+1 points. The spline is CX, CY, or CZ continuous and defined by coefficients such that:
\[ x(i) = AX(i) + BX(i) * S + CX(i) * S**2 + DX(i) * S**3 \]
\[ y(i) = AY(i) + BY(i) * S + CY(i) * S**2 + DY(i) * S**3 \]
\[ z(i) = AZ(i) + BZ(i) * S + CZ(i) * S**2 + DZ(i) * S**3 \]

and a parameter space \((T_0, T_1, \ldots, T_n)\)

where

\[ T(i) < u < T(i+1) \]
\[ S = u - T(i) \]

**Position Tolerance** — A position tolerance (true position) specifies a tolerance zone within which the feature may vary in any direction.

**Post-processor** — A phase of the translator where data is received from the Exchange Format and is converted to the Working Form.

**Pre-processor** — A phase of the translator where data is taken from the Working Form and is converted to the Exchange Format.

**Primitive Constructive Feature Representation (Generic)** — A constructive representation that is noncompound and that does not incorporate another feature. Such a representation must consist solely of overt construction information. Representation of a through hole by centerline and diameter is an example.

**Probe Blueprint** — Blueprint of Eddy current probe supplied by the probe manufacturer.

**Product Definition Data** — Those data "explicitly representing all required concepts, attributes, and relationships" normally communicated from Design throughout Manufacturing and Logistics Support. The data include both shape and nonshape information required to fully represent a component or assembly so that it can be analyzed, manufactured, inspected, and supported. They enable downstream applications, but do not include process instructions. These data are not always finalized at the design release; the manufacturing process can also add to the product model or generate derived manufacturing product models.

**Product Life Cycle** — Includes design, analysis, manufacturing, inspection, and product and logistics support of a product.
Product Model -- A computer representation of a product.

Product Support -- The function that interprets customer requests for information and can provide the technical responses to the customer in the form of technical orders and instructions.

Proprietary Part Flaw Data -- Formatted dataset containing proprietary data defining size(s), maximums, and location(s) of critical flaw(s) (dimensional and locational tolerance).

RAW.0 File -- A data file that uses a bi-cubic patch surface representation to define the surfaces of an airfoil.

Ready Status -- Go/No-Go decision.

Relation -- A logical association between entities.\(^3\)

Remount Decision -- Decision to remount an engine disk.

Replicate Feature Representation (Generic) -- A description of a feature as being identical to another feature except for location. Mathematically, a replicate feature representation consists of the identification of another (necessarily constructive) feature plus a transformation.

Robot Initialization Parameters -- A set of nondestructive testing parameters used to initialize the robot on an Eddy current or ultrasonic cell.

Rotational Sweep -- A sweep in which the swept curve is rotated about a line (the "centerline" of the sweep).

Ruled Surface (Generic) -- A surface defined by a linear blend of two curves.

Run System -- The Translator subpackage which provides the communication interface between the user and the pre/Post-processors.

Run-Time Subschema -- A subset of the data dictionary information used at run-time by the access software to provide field data and check data.

Scan Plan -- Instructions that drive an inspection; these include inspection area geometry, ordered inspection path points, inspection probe selection, inspection path for each probe, mechanical commands that allow mechanical manipulator positioning, instrument setting, and all the variables needed for signal processing and flaw data acquisition during inspection.

2-15
Scan Plan Specifications -- Standards and procedures used in creating Eddy current and ultrasonic scan plans for the RFC system.


Shape -- The physical geometry of a mechanical part, as distinguished from a computer description of that geometry. Where the difference is significant, the attitude is taken that shape is nominal or basic, with shape variations of tolerances grafted thereon.

Shape Data -- Include the geometric, topological description of a product along with the associated dimensional tolerances and feature descriptions.

Single Spatial Probe/Transducer Path -- The starting and ending location of a single probe movement.

Size Tolerance -- Size tolerances specify the allowable variation in size-of-model features, independent of location. Size tolerances include circumferential, rectilinear size, and angular size.

Solid Geometric Model (Shape Representation) -- A computer description of shape. The description may be partial in the sense that not all aspects of part shape are indicated. For example, a body of revolution representation of a turned part may not describe the nonaxisymmetric aspects of part geometry. A solid model must be complete and unambiguous in the sense that it describes a single volume in 3-D space.

Solid Modeling -- The creation of an unambiguous and complete representation of the size and shape of an object.

Source Code -- A computer program written in some language which is processed to produce machine code.

Spline -- A piecewise polynomial of order K, having continuity up to order K-1 at the segment joints.

Squirter Blueprint -- Blueprint of the squirter head that houses the ultrasonic transducer.

Subface -- A subface is a bounded portion of a face. It is defined by an underlying face, exactly one periphery closed curve and zero, one, or more internal closed curves that represent cutouts or holes in the region. The internal closed curve must not touch or intersect each other or the periphery closed curve and must be entirely contained within the periphery closed curve.
Surface Flaw -- A surface anomaly.

Surface Flaw Major Characteristic -- A vector determined by an agreed upon method.

Example: A vector representing the major axis of the minimum elliptical envelope encompassing the anomaly in the plane of the surface.

Surface Flaw Tolerance -- A unique combination of:

(a) Surface flaw orientation range.
(b) Serviceable surface flaw tolerance limits.
(c) Repairable surface flaw tolerance limits.

Surface Flaw Tolerance Limit -- A unique combination of:

(a) Maximum length.
(b) Maximum width.
(c) Maximum depth.

Sweep Surface -- Surfaces formed by extruding or revolving a planar profile in space.

Syntax -- Grammar: A set of rules for forming meaningful phrases and sentences from words in a vocabulary.

System Computer -- VAX 11/780 and supporting peripheral hardware.

System Constraints -- Those hardware and software environmental constraints which will be imposed upon the PDDI Access Software that will limit its implementation and application. An example of such constraints might be the particular compiler used to compile the PDDI Access Software package.

To-Be -- The future condition possible, given a proposed capability.

Tolerance (Generic) -- The total amount by which something may vary. For mechanical product definition, tolerances can be shape tolerances, weight tolerances, finish tolerances and so on. In the context of GMAP, the term "tolerance" used alone implies shape tolerance. Other forms of tolerance (nonshape) are explicitly stated, for example, "finish tolerance." In a GMAP product model, tolerances occur without dimensions. As in the Product Definition Data Interface Program, model dimensions are implicit in the model geometry. Therefore, application of a tolerance implies a specific underlying dimension or geometric condition.

Topology -- A data structure that assembles geometric entities (points, curves, surfaces) into a solid geometric model.
Transducer Blueprint — Blueprint of ultrasonic transducer supplied by the transducer manufacturer.

Transfer Data — The data required to make an exchange of data between systems (i.e., delimiters, record counts, record length, entity counts, numeric precision).

Translator — A software MECHANISM that is used for passing data between the Exchange Format and Working Form of the PDD.

Ultrasonic Cell — Hardware used to perform ultrasonic inspection operation (internal flaws).

Ultrasonic Inspection — An inspection method used to detect surface flaws on a disk. It uses ultrasonic waves through a stream of water to send and collect signals concerning an area targeted for inspection.

Ultrasonic Scan Plan — Interpreter code program controlling the ultrasonic inspection of a particular geometry.

Unbounded Geometry — Geometry represented parametrically, without limits, usually by coefficients to a defining equation.

Unigraphics (UG) — A computer graphics system.

User Function (UFUNC) — An interface to the UG database.

Working Form — Product definition data information in machine-dependent data formats; an a memory resident network model.

Zone — A physical area of the disk composed of zone components.

Zone Component — A subface, face, or feature that constitutes a zone or element of a zone.

NOTES:


3 Ibid., p. 214.

4 Ibid., p. 211. 

2-18
2.2.2 Acronyms Used In GMAP

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADB</td>
<td>Application Data Block (also referred to as Attribute Data Block).</td>
</tr>
<tr>
<td>AIMS</td>
<td>Automated IDEF Methodology System.</td>
</tr>
<tr>
<td>ANSI</td>
<td>American National Standards Institute.</td>
</tr>
<tr>
<td>ANT</td>
<td>Abstract of New Technology.</td>
</tr>
<tr>
<td>APT</td>
<td>Automatically Programmed Tools.</td>
</tr>
<tr>
<td>ATP</td>
<td>Automation Technology Products.</td>
</tr>
<tr>
<td>BOM</td>
<td>Bill of Materials.</td>
</tr>
<tr>
<td>BOR</td>
<td>Body of Revolution.</td>
</tr>
<tr>
<td>BPI</td>
<td>Bits per Inch.</td>
</tr>
<tr>
<td>BREP</td>
<td>Boundary Representation.</td>
</tr>
<tr>
<td>CAD</td>
<td>Computer Aided Design.</td>
</tr>
<tr>
<td>CAE</td>
<td>Computer Aided Engineering.</td>
</tr>
<tr>
<td>CAEDS</td>
<td>Computer Aided Engineering Design System.</td>
</tr>
<tr>
<td>CALS</td>
<td>Computer Aided Acquisition and Logistics Support.</td>
</tr>
<tr>
<td>CAM</td>
<td>Computer Aided Manufacturing.</td>
</tr>
<tr>
<td>CAM-I</td>
<td>Computer Aided Manufacturing--International.</td>
</tr>
<tr>
<td>CAPP</td>
<td>Computer Aided Process Planning.</td>
</tr>
<tr>
<td>CAS</td>
<td>Cooled Airfoil System.</td>
</tr>
<tr>
<td>CDM</td>
<td>Common Data Model.</td>
</tr>
<tr>
<td>CDR</td>
<td>Critical Design Review.</td>
</tr>
<tr>
<td>CDT</td>
<td>Component Design Technology.</td>
</tr>
<tr>
<td>CFSR</td>
<td>Contract Fund Status Report.</td>
</tr>
<tr>
<td>CI</td>
<td>Configuration Item.</td>
</tr>
<tr>
<td>CIM</td>
<td>Computer Integrated Manufacturing.</td>
</tr>
<tr>
<td>CLIST</td>
<td>IBM command list.</td>
</tr>
<tr>
<td>CM</td>
<td>Configuration Management.</td>
</tr>
<tr>
<td>CMM</td>
<td>Coordinate Measuring Machine.</td>
</tr>
<tr>
<td>C/SSR</td>
<td>Cost/Schedule Status Report.</td>
</tr>
<tr>
<td>CWBS</td>
<td>Contract Work Breakdown Structure.</td>
</tr>
<tr>
<td>DBMS</td>
<td>Data Base Management System.</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>DCL</td>
<td>DEC Command Language.</td>
</tr>
<tr>
<td>DDL</td>
<td>Data Definition Language.</td>
</tr>
<tr>
<td>DEA</td>
<td>Digital Equipment Automation.</td>
</tr>
<tr>
<td>DEC</td>
<td>Digital Equipment Corporation.</td>
</tr>
<tr>
<td>DESO</td>
<td>(ICAM) Architecture of Design.</td>
</tr>
<tr>
<td>DJR</td>
<td>Design Job Request; Drafting Job Request.</td>
</tr>
<tr>
<td>DoD</td>
<td>Department of Defense.</td>
</tr>
<tr>
<td>DS</td>
<td>Design Specification.</td>
</tr>
<tr>
<td>DSM</td>
<td>Design Substantiation Memo.</td>
</tr>
<tr>
<td>EBCDIC</td>
<td>Extended Binary Coded Decimal Interchange Code (IBM character set).</td>
</tr>
<tr>
<td>EC</td>
<td>Eddy Current.</td>
</tr>
<tr>
<td>ECO</td>
<td>Engineering Change Order.</td>
</tr>
<tr>
<td>EDM</td>
<td>Electrical Discharge Machining.</td>
</tr>
<tr>
<td>EF</td>
<td>Exchange Format.</td>
</tr>
<tr>
<td>EII</td>
<td>Engineering Information Index.</td>
</tr>
<tr>
<td>EMD</td>
<td>Engineering Master Drawing.</td>
</tr>
<tr>
<td>EPCS</td>
<td>Engine Product Configuration Support.</td>
</tr>
<tr>
<td>ESA</td>
<td>Engineering Source Approval.</td>
</tr>
<tr>
<td>ESP</td>
<td>Experimental Solids Proposal.</td>
</tr>
<tr>
<td>FEDD</td>
<td>For Early Domestic Dissemination.</td>
</tr>
<tr>
<td>FEM</td>
<td>Finite-Element Modeling.</td>
</tr>
<tr>
<td>FOF</td>
<td>Factory of the Future.</td>
</tr>
<tr>
<td>FOS</td>
<td>Feature of Size.</td>
</tr>
<tr>
<td>FPIM</td>
<td>Fluorescent Penetrant Inspection Module.</td>
</tr>
<tr>
<td>FSCM</td>
<td>Federal Supply Code for Manufacturers.</td>
</tr>
<tr>
<td>GE</td>
<td>General Electric.</td>
</tr>
<tr>
<td>GMAP</td>
<td>Geometric Modeling Applications Interface Program.</td>
</tr>
<tr>
<td>GSE</td>
<td>Ground Support Equipment.</td>
</tr>
<tr>
<td>HCF</td>
<td>High-Cycle Fatigue.</td>
</tr>
<tr>
<td>IBIS</td>
<td>Integrated Blade Inspection System.</td>
</tr>
<tr>
<td>IBM</td>
<td>International Business Machines.</td>
</tr>
</tbody>
</table>
ICOM — Input/Control/Output/Mechanism.
ICS — Information Computer System.
IDEF — ICAM Definition.
IDEFØ — IDEF Function Modeling.
IDEF1 — IDEF Information Modeling.
IDEF1X — IDEF Extended Information Modeling.
IDEF2 — IDEF Dynamics Modeling.
IDSS — Integrated Decision Support System.
IEEE — Institute of Electrical and Electronics Engineers.
IEN — Internal Engineering Notice.
IFS — Interface Specification.
IIS — Integrated Information Support System.
ILC — Improved Life Core.
IMS — Information Management System.
IPGS — (IBIS) Inspection Plan Generation System.
IRB — Industry Review Board.
IRIM — Infrared Inspection Module.
ISO — International Standards Organization.
ITA — Intelligent Task Automation.
ITI — International TechneGroup Incorporated.
LCF — Low-Cycle Fatigue.
MAS — Model Access Software.
MFCØ — (ICAM) Architecture of Manufacturing.
MRP — Materials Requirements Planning.
NBS — National Bureau of Standards.
M/C — Numerical Control.
NDE — Nondestructive Evaluation.

2-21
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NDML</td>
<td>Neutral Data Manipulation Language.</td>
</tr>
<tr>
<td>NDT</td>
<td>Nondestructive Test.</td>
</tr>
<tr>
<td>NTSB</td>
<td>National Transportation Safety Board.</td>
</tr>
<tr>
<td>NVI</td>
<td>Name/Value Interface.</td>
</tr>
<tr>
<td>OGP</td>
<td>Optical Gaging Products, Inc.</td>
</tr>
<tr>
<td>PD</td>
<td>Product Data.</td>
</tr>
<tr>
<td>PDD</td>
<td>Product Definition Data.</td>
</tr>
<tr>
<td>PDDI</td>
<td>Product Definition Data Interface Program.</td>
</tr>
<tr>
<td>PDES</td>
<td>Product Data Exchange Specification.</td>
</tr>
<tr>
<td>PDL</td>
<td>Program Design Language.</td>
</tr>
<tr>
<td>PED</td>
<td>Preliminary Engine Design.</td>
</tr>
<tr>
<td>PI</td>
<td>Principal Investigator.</td>
</tr>
<tr>
<td>PID</td>
<td>PDDI Interim Database.</td>
</tr>
<tr>
<td>PIES</td>
<td>Product Information Exchange System.</td>
</tr>
<tr>
<td>PMP/PMS</td>
<td>Program Management Plan/Project Master Schedule.</td>
</tr>
<tr>
<td>PROCAP</td>
<td>Process Capability.</td>
</tr>
<tr>
<td>PS</td>
<td>Product Specification.</td>
</tr>
<tr>
<td>RFC</td>
<td>Retirement for Cause.</td>
</tr>
<tr>
<td>RPM</td>
<td>Revolutions per Minute.</td>
</tr>
<tr>
<td>SA-ALC</td>
<td>San Antonio-Air Logistics Center.</td>
</tr>
<tr>
<td>SD</td>
<td>Scoping Document.</td>
</tr>
<tr>
<td>SDL</td>
<td>Source Data List.</td>
</tr>
<tr>
<td>SDS</td>
<td>System Design Specification.</td>
</tr>
<tr>
<td>SL</td>
<td>Salvage Layout.</td>
</tr>
<tr>
<td>SML</td>
<td>Source Material Log.</td>
</tr>
<tr>
<td>SOA</td>
<td>State-of-the-Art (Survey).</td>
</tr>
<tr>
<td>SOR</td>
<td>Surface of Revolution.</td>
</tr>
<tr>
<td>SPC</td>
<td>Statistical Process Control.</td>
</tr>
<tr>
<td>SPF</td>
<td>System Panel Facility.</td>
</tr>
<tr>
<td>SQA</td>
<td>Software Quality Assurance.</td>
</tr>
</tbody>
</table>

2-22
SQAP -- Software Quality Assurance Plan.
SRD -- System Requirements Document.
SRL -- Systems Research Laboratories.
SS -- System Specification.
STEP -- Standard for the Exchange of Product Model Data.
STP -- System Test Plan.
TCTO -- Time Compliance Technical Order.
TD -- Technical Data.
TDCR -- Turbine Design Cost Reduction.
TDR -- Tool Design Request.
TechMod -- Technology Modernization.
TO -- Technical Order.
TOP -- Technical and Office Protocol.
TSO -- Time-Sharing Option (IBM term).
UFUNC -- User Function.
UG -- Unigraphics.
UGFM -- Unigraphics File Manager.
USA -- Unified System for Airfoils.
USAF -- United States Air Force.
UTC -- United Technologies Corporation.
UTP -- Unit Test Plan.
UTR -- Unit Test Report.
UTRC -- United Technologies Research Center.
VAX -- Virtual Architecture Extended.
VMS -- Virtual Memory System.
WBS -- Work Breakdown Structure.
WF -- Working Form.
WPAFB -- Wright-Patterson Air Force Base.
XIM -- X-Ray Inspection Module.
SECTION 3
SYSTEM OVERVIEW

3.1 System Architecture

The purpose of the GMAP/Product Definition Data Interface (PDDI) Software System is to provide a prototype for the communication of complete Product Definition Data (PDD) between dissimilar CAD/CAM Systems. This system will serve as the information interface between engineering and manufacturing functions. As shown in Figure 3-1, it is composed of the Conceptual Schema, Schema Manager, Exchange Format (EF), the System Translator, and Model Access Software (MAS) with Name Value Interface (NVI).

The Conceptual Schema is a data dictionary that defines the data needed to define a CAD/CAM model. The Schema Manager is a software tool that will be used to manage all aspects of the creation and interrogation of the Conceptual Schema, and will be used to generate a physical schema. The EF is a neutral physical sequential format for passing data between dissimilar systems. The System Translator is the software mechanism for passing this data between the EF and the Working Form (WF) of the PDD. The MAS is a set of callable utility programs that will allow applications to manipulate and query PDD WF models. The NVI frees applications programmers from the need to be concerned with the physical location of attribute values for entities within the WF.

3.1.1 System Interfaces

The GMAP/PDDI software must interface with the computer system on which it is installed, the local (native) CAD/CAM database, the EF, the WF, and the user (application). It does this via MAS, the System Translator and local (native) developed software packages.

3.1.2 System Environment

The GMAP/PDDI system was developed in the following computing environment:

Computer/Operating System

IBM 43XX/MVS with TSO and associated tape drives, disk drives and terminals.

DEC VAX 11/780 VMS with associated tape drives, disk drives and terminals.
Figure 3-1. GMAP/PDDI System Architecture

Storage (Core) Requirements

The minimum core requirements for the PDDI software and database is 1.0M plus the size of the model. (The PDDI Machined Rib model required 0.57M)

Compilers

IBM-PASCAL/VS Release 2.2
DEC-PASCAL V3.3, FORTRAN 77 V4.4
3.2 Schema Manager

The Schema Manager enables the data administrator to create and maintain entity definitions in a Conceptual Schema model, analyze the defined entities, and generate physical schema from the Conceptual Schema.

3.2.1 Physical Schemas

The WF physical schema is determined through a data dictionary or PASCAL include files. The EF physical schema is defined by the Conceptual Schema and the specification for the neutral file format.

3.3 EF

The EF is a neutral data format. This physical, sequential format is used to for passing data between dissimilar CAD/CAM systems.

3.4 System Translator

The System Translator is the software package used to format PDD for transmission between different CAD/CAM systems. The Translator has a "Preprocessor" which collects data from the sending system and formulates it into an EF file; and a "Postprocessor" which collects the EF file and formulates it into the receiving system internal WF.

3.5 Model Access Software

The MAS is a set of PASCAL procedures that maintains the physical structure of related user data in computer memory. This user data is referred to as the WF model. The MAS provides an interface to the WF model for application programs to create, relate, and access elements of user data.

The application programs are independent of the physical structure of the stored data elements. This independence ensures that as different structure techniques are implemented, the application programs need not change.
3.5.1 Data Items

The MAS manages two types of data: entities and lists. An entity is an element of data supplied by the application to be stored in the WF. A list is a collection of entity keys. A list is a collection of entity keys created by the application in the WF.

3.5.1.1 Entity

An entity is the principle data item managed by the MAS, and is:

- Defined by the conceptual schema in the application creating the entity
- Accessed by a unique key return from the create entity function
- A node in the WF structure containing an Attribute Data Block (ADB), and references to other entities in Constituent Relationships and/or User Relationships

ATTRIBUTE DATA BLOCK

An ADB is a collection of data embedded in a single contiguous block of memory. Individual pieces of data within an ADB are called attributes. MAS manages only the first three items in the structure of an ADB. These three attributes, KIND, LENGTH, and SYSUSE, are required in every entity. Each attribute is described below:

KIND - Must be the first item defined in the ADB. The KIND defines the entity type code. This code cannot be changed.

LENGTH - Must be the second item defined in the ADB. The LENGTH defines the number of bytes in the ADB including KIND, LENGTH, and SYSUSE.

SYSUSE - One full word of system use data reserved for internal purposes. These data are never used by the application, and should never be inspected or modified.

NOTE: All other attribute data in the ADB is managed by the application program.

The MAS allows the structuring of the user data. The entities can be related in user/constituent order. An entity may be related to multiple user entities, creating a network structure in the WF. An entity may also contain multiple constituent entities.
CONSTITUENT RELATIONSHIP

A constituent entity is used in the definition of the user entity. Inclusive constituents of an entity encompass all descendants, their descendants, and so forth until there are no more descendants. For example in Figure 3-2, Point 0 (P0) and Point 1 (P1) are constituents of Line 1.

\begin{verbatim}
LINE = ENTITY(5008);
  IDENT : KEY T_IDENT;
  DISPLA : T_DISPLAY;
  P0 : POINT;
  P1 : POINT;
END_ENTITY;
\end{verbatim}

Figure 3-2. LINE: An Entity With Constituents
USER RELATIONSHIPS

A user entity uses constituent entities in its definition. Inclusive users of an entity include all ancestors, their ancestors, and so forth until there are no more ancestors. For example in Figure 3-2, Line 1 is a user of Point 0 (P0) and Point 1 (P1).

3.5.1.2 List

A list is a collection of entity keys which is:

- Created by the application program
- Accessed by a unique list key returned from the Create List Functions
- Used by the Application to store selected entity keys for subsequent processing.

3.5.2 Interface Parameters

The MAS is a set of PASCAL routines which provides an interface to the WF model. These routines contain input and output arguments referred to here as "interface parameters." Each interface parameter has a name and a type. This information is shown as follows:

DATA-NAME:DATA-TYPE.

3.5.2.1 Data-Name Parameters

The following conventions are used to name parameters:

- Keys are named KEY1, KEY2,...KEYN.
- The ADB is named ENTDEF.
- Text parameters are named according to their purpose.
- Integer parameters are named according to their purpose.
- A return code is produced by every interface routine/operation. This parameter is a full word integer and is always named IRC. (See Appendix C for a return code list.)
3.5.2.2 **Data-Type Parameters**

Data-Type parameters may be one of the following:

**ANYKEY** - Access key of an entity or list.

**ENTBLOCK** - Entity data block definition.
- In PASCAL, probably declared as a record.
- In FORTRAN, declared as a common or dimension array.

**CHARACTER** - A single character as defined by the system.

**INTEGER** - A full word integer.

3.5.2.3 **Formal Data Types**

The following is a reference list of data-types for interface calls in this MAS document.

```
ANYKEY    = INTEGER
LISTKEY   = ANYKEY
ENTKEY    = ANYKEY
ORD_KIND  = INTEGER
EXT_RET_CODE = INTEGER
LISTPSTN  = INTEGER
LISTINDEX = INTEGER
LISTSIZE  = INTEGER
ROUTINE   = ARRAY(1...8) OF CHARACTER
NAMTYP    = ARRAY(1...6) OF CHARACTER

(ADB) ENTBLOCK   = RECORD OF
                 KIND  = ORD_KIND
                 SIZE  = INTEGER
                 SYSUSE = INTEGER
                 DATA   = (USER DEFINED)
```

The formal declarations for the MAS interface routines are maintained in the member APL TYP of the library "CAD5.CMAP.V33.MASINC"

3.5.3 **Memory Manager**

A Model Access Memory Manager was developed to replace the PASCAL run-time memory manager. It reduces the number of bytes of overhead required for free-space collection, and isolates the WF model from all other PASCAL dynamic allocations.
This memory manager is currently in the MAS package and requires no user intervention for utilization.

3.6 NVI

The NVI frees applications programmers from concern for the physical location of attributes for entities in the WF of the MAS. The NVI provides the ability to alter the physical data structure without impacting program source code and removes the need to program and maintain attribute data structures and access algorithms by applications programmers. Section 5 provides detailed information on this feature.
4.1 Introduction

The Entity Operations and List Operations sections provide the applications programmer with the interface operations needed to access the data structures passed back to the application program.

Figure 4–1 illustrates the interrelationships of the MAS interface operations shown in these sections.

Figure 4-1. MAS Interface Operations
4.2 Initialization/Deletion of the MAS Working Form (WF)

Two routines provide the interface used to initialize the MAS.

The basic initialization operation (MAINIT) creates a working model and enables MAS.

The MAKILL function is used to destroy the working model and disable MAS.

An application does not have to install a data dictionary. It can create and use entities on an ad hoc basis. If a data dictionary is not installed, the following limitations are imposed:

1. Any entity type will be permitted.

2. The interface routines will not validate any operation other than outright errors; i.e., defining an ADB with a negative length. The application is - "on its own".

3. There will be no provision for organization of entities by class.

Included with the initialization and deletion operations descriptions that follow are the error and warning messages that may be returned. Appendix C contains a complete list of these messages along with their numeric codes.
MAINIT

FUNCTION: Initialize the working model.

FORMAT: MAINIT (IRC)

INPUT:
NONE

OUTPUT:
IRC : INTEGER
The procedure return code.

DESCRIPTION: The working model will be initialized.
Model Access Software is enabled.

ERRORS:

<table>
<thead>
<tr>
<th>Message</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAS_INIT_FAILED</td>
<td>Could not create schema and its root.</td>
</tr>
<tr>
<td>MAINIT_ALREADY_DONE</td>
<td>Root already created.</td>
</tr>
<tr>
<td>NOT_ENOUGH_ROOM</td>
<td>No more core memory.</td>
</tr>
</tbody>
</table>

NOTE: Do not call MAINIT twice in succession. Use a MAKILL to delete the current environment before initializing another.
**MAKILL**

**FUNCTION:** Delete the current working model.

**FORMAT:** MAKILL (IRC)

**INPUT:**
NONE

**OUTPUT:**
IRC : INTEGER
The procedure return code.

**DESCRIPTION:** The entire working model is destroyed.
Model Access Software is disabled.

**ERRORS:** None.
4.3 Entity Operations

The basic entity operations can be categorized by the following functions:

Create
Query
Update
Delete
Activate
Application Flags

All operations performed on entity constituent lists are done by list operations, with the exception of creating an entity with constituents.

Included with the entity operations descriptions presented on the pages that follow are the error and warning messages that may be returned. Appendix C contains a complete list of these messages along with their numeric codes.

4.3.1 Create Operations

These operations allow the creation of entities in the MAS WF model. The application creates the entity attribute data block in its local memory space. This includes the fields required by MAS (KIND, LENGTH, and SYSUSE) as well as the attributes defined by the application.

The create routines are shown in Table 4-1, and the following pages.

**TABLE 4-1**

CREATE ROUTINES

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>ROUTINE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create an entity.</td>
<td>MAECR</td>
</tr>
<tr>
<td>Create an entity with a constituent list of specified size.</td>
<td>MAECRN</td>
</tr>
</tbody>
</table>

4-5
MAECR

FUNCTION: Create an entity.

FORMAT: 

MAECR(ENTDEF,KEY1,KEY2,IRC)

INPUT:

ENTDEF : ENTBLOCK
The application data structure which contains the entity definition.

KEY1 : ANYKEY
The entity or list of entities to be made constituents of the entity being created.

OUTPUT:

KEY2 : ENTKEY
The key of the newly created entity.

IRC : INTEGER5
The return code.

DESCRIPTION: The entity is added to the model. Constituent entities are connected to the entity. If KEY1 is an entity key, then only that entity will become a constituent. If KEY1 is an application list key, then all entities in the application list will become constituents.

KEY1 may be nil if the entity being created is to have no constituents (a full word integer zero can be used as a nil key).

NOTE: The application is responsible for the format of the ADB data after the first three items (KIND, SIZE, SYSUSE). The KIND is a positive integer. The length is a positive integer representing the length of the ADB (including the three items required by MAS) in bytes.

The possible return code values are:

0 = Success:  
7 = Failure: Invalid KIND value
10 = Failure: Invalid KEY1
39 = Failure: No more core memory.

No entity is created for the return codes of failure (7, 10, 39).
MAECRN

FUNCTION: Create an entity with a constituent list of a given size.

FORMAT: MAECRN(ENTDEF,KEYC,KEYE,NUM,IRC)

INPUT:
ENTDEF : ENTDATA
The application data block that contains the entity definition.

KEYC : ANYKEY
The entity or list of entities to be made constituents of the entity being created.

NUM : INTEGER
The size of the constituent list of the entity being created.

OUTPUT:
KEYE : ENTKEY
The key of the newly created entity.

RC : INTEGER
The return code.

DESCRIPTION: The entity is added to the model. A constituent list of the given size is created and the constituent entities are added.

A nil key may be used if the entity being created is to have no constituents at this time. A full word integer zero can be used as a nil key.

The possible return code values are:

0 = Success
7 = Failure: Illegal KIND value or model corrupted.
9 = Failure: Constituent key is not an entity or an application list.
38, 39, 40 = Failure: No more core memory.

No entity is created for failures (7, 9, 38, 39, 40).
4.3.2 Query Operations

These operations are used to retrieve information for a specified entity and load it into the application memory area.

The query routines are shown in the Table 4-2, and the following pages.

TABLE 4-2
QUERY ROUTINES

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>ROUTINE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get the KIND value of a specific entity.</td>
<td>MAEGKN</td>
</tr>
<tr>
<td>Get the ADB of a specific entity.</td>
<td>MAEGTK</td>
</tr>
<tr>
<td>Determine the number of users.</td>
<td>MAEUSR</td>
</tr>
</tbody>
</table>
MAEGKN

FUNCTION: Get the KIND value of a specific entity.

FORMAT: MAEGKN(KEY1,KIND,IRC)

INPUT:
   KEY1 : ENTKEY
   The entity whose kind is to be retrieved.

OUTPUT:
   KIND : INTEGER
   The KIND value of the specified entity.
   IRC : INTEGER
   The return code.

DESCRIPTION: The stored ADB is located. The KIND value in the ADB is retrieved.

The possible return code values are:

0 = Success
18 = Failure: KEY1 is nil or not an entity.
FUNCTION: Get the ADB of a specific entity.

FORMAT: MAEGTK(KEY1,ENTDEF,IRC)

INPUT:  
KEY1 : ENTKEY  
The key of the entity whose ADB is to be copied.

OUTPUT:  
ENTDEF : ENTBLOCK  
The ADB to receive the stored entity.
IRC : INTEGER  
The return code.

DESCRIPTION: The stored ADB is located and copied into the application data block. If KEY1 is nil, then a nil KIND and a zero length is returned.

The possible return code values are:

0 = Success
18 = Failure: KEY1 is nil or not an entity.
MAEUSR

FUNCTION: Determine the number of users for an entity or application list of entities.

FORMAT: MAEUSR(KEY1,UEXIST,IRC)

INPUT:
  KEY1 : AMYKEY
  The entity or application list of entities whose users will be counted.

OUTPUT:
  UEXIST : INTEGER
  The value number of users.

  IRC : INTEGER
  The return code.

DESCRIPTION: KEY1 may be either an entity key or an application list key. If KEY1 is an entity, the number of users of the entity is returned. If KEY1 is an application list, the number of direct users of the entities on the list is returned.

The possible return code values are:

  0 = Success
  17 = Failure KEY1 is not a valid ENTKEY OR LISTKEY.
  18 = Failure KEY1 is nil.

4-11
4.3.3 Update Operations

Update operations, presented in Table 4-3, and the following pages, are used to update the ADB for specified entities. In general, the application should use the MAEGIK function to get the ADB before the update function is used.

TABLE 4-3

UPDATE OPERATIONS

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>ROUTINE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Update the attribute data block of an entity.</td>
<td>MAEUD</td>
</tr>
</tbody>
</table>
MAEUD

FUNCTION: Update the attribute data block of an entity.

FORMAT: MAEUD(KEY1,ENTDEF,IRC)

INPUT:
   KEY1 : ENTKEY
       The key of the entity to be updated.

   ENTDEF : ADB
       The ADB supplying the update values.

OUTPUT:
   IRC : INTEGER
       The return code.

DESCRIPTION: The ADB of KEY1 will be updated with the given ADB values.
   The value of KIND must agree with the working form copy.
   Otherwise, an error will result. If the length is greater than the current length, then a new ADB will
   be created with more space.

The possible return code values are:

   0 = Success
   1 = Failure Kind or given key is undefined.
   6 = Failure Cannot update entity.
   18 = Failure KEY1 is nil.
   38, 39, 40 = Failure No more core memory.
4.3.4 Delete Operations

These operations address how you delete entities from the MAS WF model. The entities in the working model currently are grouped into the following classifications:

- Dependent
- Support
- Primary
- Secondary.

Delete rules have been established for the entities in these classifications. For a new entity kind, the default classification is "Dependent" unless it is otherwise defined.

Delete Rules - The delete rules, shown in Table 4-4, apply to the constituent relationships with which entities are defined. They determine whether a constituent entity can be deleted by checking each of its user entities. For example, the delete rules applied to entity (A) in relation to a specific user entity (B) may be different than the delete rules for that same entity (A) in relation to another specific user entity (C).

The action taken for the IDBMAS delete classifications are determined by the combinations of yesno (YN) answers to the following conditions, posed as questions:

1. Can this constituent entity be deleted from a specific user entity?
2. Does the deletion of this (constituent) entity cause deletion of a specific user?
3. Does deletion of a specific user cause deletion of this entity (constituent)?
### TABLE 4-4

**DELETE RULES**

<table>
<thead>
<tr>
<th>CONDITION</th>
<th>(1)</th>
<th>(2)</th>
<th>(3)</th>
<th>DELETE CLASSIFICATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Dependent</td>
</tr>
<tr>
<td>N</td>
<td>N</td>
<td>Y</td>
<td></td>
<td>Support</td>
</tr>
<tr>
<td>N</td>
<td>Y</td>
<td>N</td>
<td></td>
<td>Primary</td>
</tr>
<tr>
<td>Y</td>
<td>N</td>
<td>N</td>
<td></td>
<td>Secondary</td>
</tr>
</tbody>
</table>

The delete classifications are defined as follows:

- **Dependent** - Constituent entity cannot be deleted because the user entity is dependent on its existence. The user entity may be deleted without deleting the constituent entity.

- **Support** - Constituent entity cannot be deleted because the user entity is dependent on its existence. The user entity may be deleted; however, the constituent entity will also be deleted unless another user entity does not permit the deletion of the constituent entity.

- **Primary** - Constituent entity can be deleted, but only if the user entity can, and will, also be deleted. The user entity may be deleted without the constituent entity being deleted.

- **Secondary** - If the number of constituents falls below an established minimum, the constituent entity can be deleted and, if possible, the user entity will also be deleted. If the user entity cannot be deleted, none of the minimum constituents can be deleted. If the number of constituents is greater than or equal to the minimum, the constituent entity can be deleted.

Test routines are provided to return the entities or lists that would be deleted if actual delete routines were used.

**Delete Routines** - The IDB/MAS delete routines are presented in Table 4-5, and the following pages. The first two routines actually delete entities (MAED, MAEDI). The third and fourth routines test the delete function, allowing the programmer to see the results of a potential delete without modifying the stored data (MAEDT, MAEDTI).
When deleting a list of entities that includes users and constituents, the list should be ordered so that the users are processed before the constituents. The routines MALROR and MALRORI perform this function on an application list. (An entity constituent list should never be reordered.)

**TABLE 4-5**

DELETE ROUTINES

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>ROUTINE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delete an entity or list of entities.</td>
<td>MAED</td>
</tr>
<tr>
<td>Delete an entity or list of entities and the inclusive constituents.</td>
<td>MAEDI</td>
</tr>
<tr>
<td>Delete test an entity or list of entities.</td>
<td>MAEDT</td>
</tr>
<tr>
<td>Delete test an entity or list of entities and the inclusive constituents.</td>
<td>MAEDTI</td>
</tr>
</tbody>
</table>
FUNCTION: Delete an entity or list of entities.

FORMAT: MAED(KEY1,KEY2,IRC)

INPUT:
   KEY1 : ANYKEY
   The entity or list of entities to be deleted.

OUTPUT:
   KEY2 : LISTKEY
   The list of entities marked for deletion.
   IRC : INTEGER
   The return code.

DESCRIPTION: KEY1 may be either an entity key or a list key, and the order of the entities in the list may be important. KEY2 will list any entities from the KEY1 list that were not deleted. If all entities are deleted, the mark list will be empty.

The possible return code values are:

   0 = Success
   17 = Failure  KEY1 is not a valid entity key or list key.
   38, 39, 40 = Failure  No more core memory.
   42 = Failure  Delete rules defined incorrectly.
   -3 = Warning  KEY1 is nil.
   -7 = Warning  No entities marked for delete.

No mark list is created for failures or warnings.
MAEDI

FUNCTION: Delete an entity or list of entities and their inclusive constituents.

FORMAT: MAEDI(KEY1,KEY2,IRC)

INPUT:
KEY1 : ANYKEY
The entity or list of entities to be deleted.

OUTPUT:
KEY2 : LISTKEY
The list of entities marked for delete.
IRC : INTEGER
The return code.

DESCRIPTION: KEY1 may be either an entity key or a list key, and the order of the entities in the list may be important. KEY2 will list any entities from the KEY1 list that were not deleted. If all entities are deleted, the mark list will be empty.

The possible return code values are:

0 = Success
17 = Failure KEY1 is not a valid entity key or list key.
38, 39, 40 = Failure No more core memory.
42 = Failure Delete rules defined incorrectly.
-3 = Warning No entities to be tested for delete, or no entities would be deleted.
-7 = Warning No entities marked for delete.

No mark list is created for failures or warnings.
MAEDT

FUNCTION: Delete test an entity or list of entities.

FORMAT: MAEDT(KEY1,KEY2,KEY3,IRC)

INPUT:
KEY1 : ANYKEY
The entity or list of entities to be tested.

OUTPUT:
KEY2 : LISTKEY
The list containing entities that would be deleted by MAED.

KEY3 : LISTKEY
The list containing entities that would be marked by MAED.

IRC : INTEGER
The return code.

DESCRIPTION: The MAEDT delete routine simulates the output of the MAED routine without actually deleting the entities or marking them inactive.

The possible return code values are:

0 = Success
17 = Failure KEY1 is not a valid entity key or list key.
38, 39, 40 = Failure No more core memory.
42 = Failure Delete rules defined incorrectly.
-3 = Warning No entities to be tested for delete or no entities would be deleted.
-7 = Warning No entities marked for delete.
MAEDTI

FUNCTION: Delete test an entity or list of entities and their inclusive constituents.

FORMAT: MAEDTI(KEY1,KEY2,KEY3,IRC)

INPUT:
   KEY1  : ANYKEY
   The entity or list of entities to be tested.

OUTPUT:
   KEY2  : LISTKEY
   The list containing entities that would be deleted by MAEDI.
   KEY3  : LISTKEY
   The list containing entities that would be marked by MAEDI.
   IRC   : INTEGER
   The return code.

DESCRIPTION: The MAEDTI delete routine simulates the output of the MAEDI routine without actually deleting the entities or rendering them inactive.

The possible return code values are:

   0 = Success
   17 = Failure  KEY1 is not a valid entity key or list key.
   38, 39, 40 = Failure
   42 = Failure  No more core memory.
   -3 = Warning  Delete rules defined incorrectly.
   -7 = Warning  No entities to be tested for delete or no entities would be deleted.

   4-20
4.3.5 **Activate Operations**

These operations are used to activate an entity. An entity is deactivated when a delete was attempted, but was not completed because of the user's dependency condition on the entity. (See Delete Operations Section.)

The activate routines are shown in Table 4-6 and the following pages.

**TABLE 4-6**

**ACTIVATE ROUTINES**

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>ROUTINE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate an entity or list of entities.</td>
<td>MAEA</td>
</tr>
<tr>
<td>Activate an entity or list of entities and their inclusive constituents.</td>
<td>MAEAI</td>
</tr>
<tr>
<td>Find the present value of the activation setting for an entity.</td>
<td>MAEAV</td>
</tr>
</tbody>
</table>

**NOTE:**

- Activation is not the same as rejection after a delete. If an entity was deleted, then it cannot be recovered with these functions.

- Activation functions will activate any entity regardless of when or how it was made inactive.
MAEA

FUNCTION: Activate an entity or list of entities.

FORMAT: MAEA (KEY1,IRC)

INPUT:
KEY1 : ANYKEY
The entity or list of entities to be activated.

OUTPUT:
IRC : INTEGER
The return code.

DESCRIPTION: KEY1 may be either an entity key or a list key. If KEY1 is an entity key then only that entity will be activated. If KEY1 is a list key then all entities in the list will be activated.

The possible return code values are:

0 = Success
17 = Failure KEY1 is not a valid entity key of list key.
18 = Failure KEY1 is nil.
38, 39, 40 = Failure No more core memory.
MAEAI

FUNCTION: Activate an entity or list of entities and their inclusive constituents.

FORMAT: MAEAI(KEY1, IRC)

INPUT:
   KEY1  : ANYKEY
       The entity or list of entities to be activated.

OUTPUT:
   IRC  : INTEGER
       The return code.

DESCRIPTION: KEY1 may be either an entity key or a list key. If KEY1 is an entity key then only that entity and its inclusive constituents will be activated. If KEY1 is a list key then all entities in the list and their inclusive constituents will be activated.

Refer to the System Overview Section for further explanation of inclusive constituents.

The possible return code values are:

   0 = Success
   17 = Failure  KEY1 is not a valid entity key or list key.
   18 = Failure  KEY1 is nil.
   38, 39, 40 = Failure  No more core memory.
MAEAV

FUNCTION: Find the present value of the activation setting for an entity.

FORMAT: MAEAV(KEY1,IAVAL,IRC)

INPUT:
   KEY1 : ENTKEY
   The entity to be examined.

OUTPUT:
   IAVAL : INTEGER
   The activation code.
   = 0  if set for delete
   = 1  if not set for delete

   IRC : INTEGER
   The return code.

DESCRIPTION: Returns the current value of the activation setting for the specified entity.

The possible return code values are:

0 = Success
18 = Failure KEY1 is nil.
38, 39, 40 = Failure No more core memory.
4.3.6 Application Flag Operations

These operations are used to query or set any application accessible flag associated with an entity.

The Application Flag routines are shown in Table 4-7, and the following pages.

TABLE 4-7

APPLICATION FLAG OPERATIONS

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>ROUTINE</th>
</tr>
</thead>
<tbody>
<tr>
<td>For all entities in the model, reset the specified flag.</td>
<td>MAERST</td>
</tr>
<tr>
<td>For all entities in the model, reset the process flag and the application flag.</td>
<td>MABRST</td>
</tr>
<tr>
<td>For an entity or list of entities, update the specified flag.</td>
<td>MAUPDT</td>
</tr>
<tr>
<td>For the constituents of an entity or list of entities, update the specified flag.</td>
<td>MACPDT</td>
</tr>
<tr>
<td>For the inclusive constituents of an entity or list of entities, update the process flag.</td>
<td>MAESCI</td>
</tr>
<tr>
<td>For an entity, query the specified flag.</td>
<td>MAQURY</td>
</tr>
<tr>
<td>Determine whether the user compresses with its constituent.</td>
<td>MAECQY</td>
</tr>
<tr>
<td>Create a list of constituents with which the input entity compresses.</td>
<td>MAECMP</td>
</tr>
</tbody>
</table>
TABLE 4-7 (contd.)

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>ROUTINE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reset Process Flag for all entities in the model.</td>
<td>MAESWA</td>
</tr>
<tr>
<td>Set the Process Flag in an entity or list of entities.</td>
<td>MAESWT</td>
</tr>
<tr>
<td>Find the Process Flag setting of an entity.</td>
<td>MAESVL</td>
</tr>
</tbody>
</table>
MAERST

FUNCTION: Reset given application accessible flag in all entities in the model.

FORMAT: MAERST(FLAGNAME,IRC)

INPUT:

FLAGNAME : NAMTYP
The name of the flag to be reset in all entities in the model. Can have the following values:

1) /MRDFlG' activation flag
2) /PRCFlG' process flag
3) /ABSFlG' absent/present flag
4) /APLFlG' application flag

OUTPUT:

IRC : INTEGER
The return code.

DESCRIPTION: Determine what flag is to be reset in every entity in the model. Resets that flag to /off'.

The possible return code values are:

0 = Success
34 = Failure
35 = Failure
38, 39, 40 = Failure

Invalid flag name.
No model established.
No more core memory.
MABRST

FUNCTION: Reset the process and application flags on each entity in the working form model.

FORMAT: MABRST(IRC)

INPUT:
NONE

OUTPUT:
IRC : INTEGER
The return code.

DESCRIPTION: For each entity in the working form model, the process and application SYSUSE flags are turned off.

The possible return code values are:

0 = Success
35 = Failure No model has been established.
MAUPDT

FUNCTION: Update the value of a given application accessible flag for an entity or list of entities.

FORMAT: MAUPDT(KEY1,FLGNAME,FLGVAL,IRC)

INPUT:
KEY1 : ANYKEY
The entity or list of entities whose specified flag value will be updated.

FLAGNAME : IAMTYP
The name of the flag to be updated. Can have the following values:
1) /MRDFLG' activation flag
2) /PRCFLG' process flag
3) /ABSFLG' absent/present flag
4) /APLFLG' application flag

FLGVAL : INTEGER
The value of the specified flag to be used when updating.
0 = false
1 = true

IRC : INTEGER
The return code.

DESCRIPTION: Determine which flag is to be updated, and replace that value.
The possible return code values are:

0 = Success
17 = Failure KEY1 is nil or not an entity.
34 = Failure Invalid flag name.
38, 39, 40 = Failure No more core memory.
MACPDT

FUNCTION: Update a specified SYSUSE flag value for the constituents of an entity or a list of entities.

FORMAT: MACPDT(KEY1,FLGNAME,FLGVAL,IRC)

INPUT:
KEY1 : ANYKEY
The key of an entity or list of entities whose constituents will be updated.

FLGNAME : RAMTYP
The name of the flag to be updated. It can have the following values:
1) /MRDFLG' activation flag
2) /PRCFLG' process flag
3) /ABSFLG' absent/present flag
4) /APLFLG' application flag

FLGVAL : INTEGER
The value of the specified flag to be used when updating.
0 = False
1 = True

OUTPUT:
IRC : INTEGER
The return code.

DESCRIPTION: Determine what value of a flag is to be updated. Collect the constituent entities to be updated. Update the entities.

The possible return code values are:

0 = Success
17 = Failure KEY1 is not a valid entity key or list key.
18 = Failure KEY1 is nil.
34 = Failure Undefined flag name.
38, 39, 40 = Failure No more core memory.
-6 = Warning The entity or list of entities had no constituents.
MAESCI

FUNCTION: Set or reset the process flag for the inclusive constituents of an entity or a list of entities.

FORMAT: MAESCI(KEY1,ISWT,IRC)

INPUT:
KEY1 : ANYKEY
The key to an entity or list of entities.
ISWT : INTEGER
The ordinal value of true or false.

OUTPUT:
IRC : INTEGER
The return code.

DESCRIPTION: Given a valid key, the inclusive constituents of an entity or list of entities are collected. Each collected constituent entity's process flag is updated with the given value.

The possible return code values are:

0 = Success
17 = Failure
18 = Failure
38, 39, 40 = Failure
-6 = Warning

KEY1 is not a valid entity key or list key.
KEY1 is nil.
No more core memory exist.
The entity or list of entities had no constituents.
FUNCTION: Determine the value of a given application accessible flag for the entity.

FORMAT: MAQURY(KEY1,FLAGNAME,FLGVAL,IRC)

INPUT:
KEY1 : ENTKEY
The entity whose specified flag value is to be retrieved.

FLAGNAME : NAMTYP
The name of the flag to be retrieved. Can have the following values:
1) /MRDFLG' activation flag
2) /PRCFLG' process flag
3) /ABSFILG' absent/present flag
4) /APLFLG' application flag

OUTPUT:
FLGVAL : INTEGER
The value of the specified flag.
0 = false
1 = true

IRC : INTEGER
The return code.

DESCRIPTION: Determine which flag is to be retrieved, and return that value.

The possible return code values are:
0 = Success
17 = Failure KEY1 is nil or not an entity.
34 = Failure Invalid flag name.
38, 39, 40 = Failure No more core memory.
**MAECQY**

**FUNCTION:** Determine whether the user compresses with its constituent.

**FORMAT:** MAECQY(KEY1, KEY2, CMPFLG, IRC)

**INPUT:**
- KEY1 : ENTKEY
  Key of the entity whose constituent is to be checked.
- KEY2 : ENTKEY
  Key of the constituent whose rule is to be checked.

**OUTPUT:**
- CMPFLG : INTEGER
  Value of the user's compress rule in relation to its constituent.
  - 1 = true
  - 0 = false
- IRC : INTEGER
  The return code.

**DESCRIPTION:** Query constituent compress rule to its user.

The possible return code values are:

- 0 = Success
- 17 = Failure
  KEY1 is not a valid entity key or list key.
- 18 = Failure
  KEY1 is nil.
- 38, 39, 40 = Failure
  No more core memory.
- 42 = Failure
  Delete rules defined incorrectly.

No value is returned for the return codes of failure.
MAECMP

FUNCTION: Create a list of constituents with which the input entity compresses.

FORMAT: MAECMP(KEY1,KEY2,IRC)

INPUT:
   KEY1 : ENTKEY
   Key of the entity that's compressibility is determined by the constituent(s).

OUTPUT:
   KEY2 : LISTKEY
   List of the constituents that cause the compression of the input entity.
   IRC  : INTEGER
   Return code

DESCRIPTION: KEY2 is initialized to nil. Each constituent whose delete rule states that the input entity will be compressed will be added to KEY2.

The possible return code values are:

   0 = Success
   17 = Failure       KEY1 is not a valid entity key or list key.
   18 = Failure       KEY1 is nil.
   38, 39, 40 = Failure
   42 = Failure       Delete rules defined incorrectly.
   -6 = Warning       The entity or list of entities had no constituents.

No list is created for failures or warnings.
MAESWA

FUNCTION: Reset Process Flag for all entities in the model.

FORMAT: MAESWA(IRC)

INPUT: NONE

OUTPUT: IRC : INTEGER
   The return code.

DESCRIPTION: The Process Flag is set to OFF in all entities in the working-form model.

The possible return code values are:

   0 = Success
   38, 39, 40 = Failure
   No more core memory.
MAESWT

FUNCTION:  Set the Process Flag in an entity or a list of entities.

FORMAT:   MAESWT(KEY1, ISWT, IRC)

INPUT:
  KEY1 : ANYKEY
         The entity or list of entities whose process flag is to be set.

  ISWT : INTEGER
         The input value of the process flag.

OUTPUT:
  IRC  : INTEGER
         The return code.

DESCRIPTION: The process flag will be set to the value specified by ISWT. If KEY1 is an entity key, then the flag in that entity will be set. If KEY1 is a list key, then the flag in all entities referenced by the list will be set. ISWT should be "1" for flag setting of true and "0" for flag setting of false.

The possible return code values are:

  0 = Success
  17 = Failure KEY1 is not a valid entity key or list key.
  18 = Failure KEY1 is nil.
  38, 39, 40 = Failure No more core memory.
MAESVL

FUNCTION: Find the Process Flag setting of an entity.

FORMAT: MAESVL(KEY1,ISET,IRC)

INPUT:
  KEY1 : KEY
  The entity for which the flag setting is wanted. This must be an entity key.

OUTPUT:
  ISET : INTEGER
  The output value of the process flag.
  IRC : INTEGER
  The return code.

DESCRIPTION: The value of the process flag for KEY1 will be returned. If the flag is true, then the value "1" will be returned. If the flag is false, then the value "0" will be returned.

The possible return code values are:

  0 = Success
  18 = Failure
  KEY1 is not a valid entity key or list key.
  38, 39, 40 = Failure
  No more core memory.
4.4 List Operations

This section explains the use of the MAS list operations. A list is a temporary internal structure that contains references to entities. Since the application can build lists that take up space in the working model, it is necessary that the applications periodically delete the lists that are no longer needed.

Many list operations will accept either a list key or an entity key as input keys. When an entity key is supplied, it is assumed that the constituent list of the entity becomes the list to be operated on.

Some operations on lists may result in the same entity being in the output list more than once. The operation (MALRDE) can be used to remove duplicate entities from the list.

Operations that create an application list automatically set the position of the list to the beginning. Once a list has been read to the end, it must be reset before the sequential read process can begin again.

When an entity is deleted, all references to it in all application lists are automatically removed and the current positions of the affected lists are adjusted to retain their original meaning.

The basic list operations can be categorized by the following functions:

Create application lists
Query application lists and constituent lists
Update application lists and constituent lists
Update application lists only
Boolean operations
Delete application lists.

Included with the list operations descriptions are the error and warning messages that may be returned. Appendix C contains a complete list of these messages along with their numeric codes.
### 4.4.1 Create Operations – Application Lists

These operations create application lists. The first two create empty lists to be updated by the user routine. The others create lists of entities based on some selection criteria.

The create routines are shown Table 4-8, and the following pages.

#### TABLE 4-8
CREATE ROUTINES

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>ROUTINE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creates an empty list.</td>
<td>MAL</td>
</tr>
<tr>
<td>Create an empty list of specified size.</td>
<td>MALN</td>
</tr>
<tr>
<td>Create a list of entities with a given KIND.</td>
<td>MALK</td>
</tr>
<tr>
<td>Create a list of entities with a given KIND that are found within another list.</td>
<td>MALKL</td>
</tr>
<tr>
<td>Makes a copy of a list.</td>
<td>MALCPY</td>
</tr>
<tr>
<td>Create an application list of constituent entity references.</td>
<td>MAEC</td>
</tr>
<tr>
<td>Create an application list of inclusive constituent entities.</td>
<td>MAECI</td>
</tr>
<tr>
<td>Create an application list of inclusive constituents by KIND.</td>
<td>MAECIK</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>ROUTINE</td>
</tr>
<tr>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>Create an application list of entities of a specified KIND taken from the constituents of an entity or from the constituents of a list of entities.</td>
<td>MALKC</td>
</tr>
<tr>
<td>Create an application list of user entity references.</td>
<td>MAEU</td>
</tr>
<tr>
<td>Create an application list of inclusive user entities.</td>
<td>MAEUI</td>
</tr>
<tr>
<td>Create an application list of inclusive users by KIND.</td>
<td>MAEUIK</td>
</tr>
<tr>
<td>Create an application list of entities of a specified KIND taken from the users of an entity or from the users of a list of entities.</td>
<td>MALKU</td>
</tr>
</tbody>
</table>
FUNCTION: Creates an empty list.

FORMAT: MAL(KEY1,IRC)

INPUT: NONE

OUTPUT:
KEY1 : LISTKEY
The key of the empty list.

IRC : INTEGER
The return code.

DESCRIPTION: An empty list is created.

The possible return code values are:

0 = Success
3 = Failure (Cannot create list.)
38, 39, 40 = Failure (No more core memory.)
FUNCTION: Create an empty list of specified size.

FORMAT: MALN(LSIZE,KEY1,IRC)

INPUT:
- LSIZE : INTEGER
  The number of entries in the list.

OUTPUT:
- KEY1 : LISTKEY
  The key of the empty list of specified size.
- IRC : INTEGER
  The return code.

DESCRIPTION: An empty application list will be created with sufficient space to accommodate LSIZE entries. All entries are initialized to nil.

The possible return code values are:

0 = Success
3 = Failure          Cannot create list.
15 = Failure         Requested a list size too large.
38, 39, 40 = Failure No more core memory.
MALK

FUNCTION: Create a list of entities with a given KIND.

FORMAT: MALK(KIND,KEY1,IRC)

INPUT:
KIND : INTEGER
A valid KIND code.

OUTPUT:
KEY1 : LISTKEY
The list of all entities of the specified KIND.
IRC : INTEGER
The return code.

DESCRIPTION: KEY1 is initialized to nil. If KIND is a valid value, all entities of the given KIND will be copied into KEY1.

The possible return code values are:

0 = Success
38, 39, 40 = Failure
-1 = Warning

No list is created for failures or warnings.
MALKL

FUNCTION: Create a list of entities with a given KIND that are found within another list.

FORMAT: MALKL(KEY1,KIND,KEY2,IRC)

INPUT:
KEY1 : ANYKEY
The entity or list of entities whose list is to be searched for the specified KIND.

KIND : INTEGER
The KIND code of an entity.

OUTPUT:
KEY2 : LISTKEY
The list that will contain all entities of the given KIND found within the list specified by KEY1.

IRC : INTEGER
The return code.

DESCRIPTION: KEY2 is initialized to nil. If KEY1 is an entity key, copy all constituents of the given kind on KEY2. If KEY1 is a list key, put all entities on the list of the given kind on KEY2.

The possible return code values are:

0 = Success
14 = Failure
17 = Failure
18 = Failure
38, 39, 40 = Failure
-1 = Warning
-6 = Warning

Model is corrupted.
KEY1 is not a valid entity key or list key.
KEY1 is nil.
No more core memory exists.
The entity or list of entities had no constituents or entities of the given kind.

No list is created for failures or warnings.
MALCPY

FUNCTION: Makes a copy of a list.

FORMAT: MALCPY(KEY1,KEY2,IRC)

INPUT:
   KEY1  : LISTKEY
   A list key whose entries will be copied.

OUTPUT:
   KEY2  : LISTKEY
   The new list that will receive a copy of KEY1.
   IRC   : INTEGER
   The return code.

DESCRIPTION: KEY2 will be created. The elements of KEY1 will be copied into KEY2.

The possible return code values are:

   0 = Success
   17 = Failure
   KEY1 is nil or not an application list.
   38, 39, 40 = Failure
   No more core memory.
FUNCTION: Create an application list of constituent entities.

FORMAT: MAEC(KEY1,KEY2,IRC)

INPUT:
KEY1 : ANYKEY
The entity or list of entities for which a list of direct constituents is wanted.

OUTPUT:
KEY2 : LISTKEY
The returned key of the application list of direct constituents.
IRC : INTEGER
The return code.

DESCRIPTION: KEY2 is created. If KEY1 is an entity key then the constituent list of KEY1 will be copied into KEY2. If KEY1 is a list key then the constituent lists of each entity will be copied into KEY2.

The possible return code values are:

0 = Success
17 = Failure KEY1 is not a valid entity key or list key.
18 = Failure KEY1 is nil.
39 = Failure No more core memory.
-6 = Warning The entity or list of entities had no constituents.

No list is created for the return codes of failure (17, 18, 39) or warning (-6), and KEY2 is nil.
MAECI

FUNCTION: Create an application list of inclusive entities.

FORMAT: MAECI(KEY1,KEY2,IRC)

INPUT:
KEY1 : ANYKEY
The entity or list of entities whose inclusive constituents are wanted.

OUTPUT:
KEY2 : LISTKEY
The returned key of the inclusive application list of constituents.
IRC : INTEGER
The return code.

DESCRIPTION: KEY2 is created. If KEY1 is an entity key, then the inclusive constituent list of KEY1 will be copied into KEY2. If KEY1 is a list key, then the inclusive constituent lists of each entity will be copied into KEY2. KEY1 is not included in KEY2. No duplicate keys will exist.

NOTE: See the System Overview Section, page 1.5.2, for further explanation of inclusive constituents.

The possible return code values are:

0 = Success
17 = Failure  KEY1 is not a valid entity key or list key.
18 = Failure  KEY1 is nil.
39 = Failure  No more core memory.
-6 = Warning  The entity or list of entities had no constituents.

No list is created for the return codes of failure (17, 18, 39) or warning (-6), and KEY2 is nil.
MAECIK

FUNCTION: Create an application list of inclusive constituents of a specified KIND.

FORMAT: MAECIK(KEY1,KIND,KEY2,IRC)

INPUT:

KEY1 : ANYKEY
The entity or list of entities whose inclusive constituents are to be searched for by specified KIND.

KIND : INTEGER
The KIND code of an entity or an entity class.

OUTPUT:

KEY2 : LISTKEY
The key of a list which will contain all entities of the specified KIND found within the inclusive constituents of KEY1.

IRC : INTEGER
The return code.

DESCRIPTION: KEY2 is initialized to nil. If KEY1 is a valid entity key, then the inclusive constituents of the specified KIND will be copied into KEY2. If KEY1 is a valid list key, then the inclusive constituents of all entities on the list of the specified KIND will be copied into KEY2. No duplicates will exist.

The possible return code values are:

0 = Success
17 = Failure KEY1 is not a valid entity key or list key.
18 = Failure KEY1 is nil.
38, 39, 40 = Failure No more core memory.
-6 = Warning The entity or list of entities had no constituents of the given KIND.

No list is created for the return codes of failure or warning.
FUNCTION: Create a list of entities of a specified kind found within the constituent list of an entity or the constituent lists of a list of entities.

FORMAT: MALKC(KEY1,KIND,KEY2,IRC)

INPUT: KEY1 : ANYKEY
The key to an entity or list of entities.

KIND : ORD_KIND
Any valid kind value.

OUTPUT: KEY2 : LISTKEY
A list of entities found within the constituent list of an entity or the constituent lists of a list of entities.

IRC : INTEGER
The return code.

DESCRIPTION: Given a valid kind value, the constituent of an entity or list of entities are collected. For each collected entity of the given kind, copy into KEY2.

The possible return code values are:

0 = Success
17 = Failure KEY1 is not a valid entity key or list key.
18 = Failure KEY1 is nil.
38, 39, 40 = Failure No more core memory.
-1 = Warning Kind value is undefined.
-6 = Warning The entity or list of entities had no constituents of the given kind.
-11 = Warning The entity or list of entities had no constituents.

No list is created for failures or warnings.
MAEU

FUNCTION: Create an application list of user entity references.

FORMAT: MAEU(KEY1,KEY2,IRC)

INPUT:
KEY1 : ANYKEY
The entity or list of entities for which a list of direct users is wanted.

OUTPUT:
KEY2 : LISTKEY
Returned key of the application list of direct users.

IRC : INTEGER
The return code.

DESCRIPTION: KEY2 is initialized to nil. If KEY1 is a valid entity key, then the user list of KEY1 will be copied into KEY2. If KEY1 is a valid list key, then the user lists of each entity will be copied into KEY2.

The possible return code values are:

0 = Success
17 = Failure KEY1 is not a valid entity key or list key.
18 = Failure KEY1 is nil.
38, 39, 40 = Failure No more core memory.
-6 = Warning The entity or list of entities had no users.

No list is created for the return codes of failure or warning.
MAEUI

FUNCTION: Create an application list of inclusive user entities.

FORMAT: MAEUI(KEY1,KEY2,IRC)

INPUT:
KEY1 : ANYKEY
The entity or list of entities whose inclusive users are wanted.

OUTPUT:
KEY2 : LISTKEY
The returned key of the inclusive application list of users.

IRC : INTEGER
The return code.

DESCRIPTION: KEY2 is initialized to nil. If KEY1 is a valid entity key, then the inclusive user list of KEY1 will be copied into KEY2. If KEY1 is a valid list key, then the inclusive user lists of each entity will be copied into KEY2. KEY1 is not included in KEY2. There will be no duplicates.

The possible return code values are:

0 = Success
17 = Failure
   KEY1 is not a valid entity key or list key.
18 = Failure
   KEY1 is nil.
38, 39, 40 = Failure
   No more core memory.
-6 = Warning
   The entity or list of entities had no users.

No list is created for failures or warnings.
MAEUIK

FUNCTION: Create an application list of inclusive users by KIND.

FORMAT: MAEUIK(KEY1,KIND,KEY2,IRC)

INPUT:
KEY1 : ANYKEY
   The entity or list of entities whose inclusive users are to be searched for by specified KIND.

KIND : INTEGER
   The KIND code of an entity or an entity class.

OUTPUT:
KEY2 : LISTKEY
   The key of a list which will contain all entities of the given KIND found within the inclusive users of KEY1.

IRC : INTEGER
   The return code.

DESCRIPTION: KEY2 is initialized to nil. If KEY1 is a valid entity key, then the inclusive users of the given KIND will be copied into KEY2. If KEY1 is a valid list key, then the inclusive users of all entities on the list of the given KIND will be copied into KEY2. No duplicates will exist.

The possible return code values are:

0 = Success
17 = Failure KEY1 is not a valid entity key or list key.
18 = Failure KEY1 is nil.
38, 39, 40 = Failure No more core memory.
-6 = Warning The entity or list of entities had no users of the given KIND.

No list is created for failures or warnings.
FUNCTION: Create a list of entities of a specified kind found within the user list of an entity or the user lists of a list of entities.

FORMAT: MALKU(KEY1,KIND,KEY2,IRC)

INPUT:
  KEY1 : ANYKEY
          The key to an entity or list of entities.
  KIND : ORD_KIND
          Any valid kind value.

OUTPUT:
  KEY2 : LISTKEY
          A list of entities found within the user list of an entity or the user lists of a list of entities.
  IRC : INTEGER
          The return code.

DESCRIPTION: Given a valid kind value, the users of an entity or list of entities are collected. For each collected entity of the given kind, copy into KEY2.

The possible return code values are:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success</td>
</tr>
<tr>
<td>17</td>
<td>Failure: KEY1 is not a valid entity key or list key.</td>
</tr>
<tr>
<td>18</td>
<td>Failure: KEY1 is nil.</td>
</tr>
<tr>
<td>38, 39, 40</td>
<td>Failure: No more core memory exists.</td>
</tr>
<tr>
<td>-1</td>
<td>Warning: Kind value undefined.</td>
</tr>
<tr>
<td>-6</td>
<td>Warning: The entity or list of entities had no users of the given KIND.</td>
</tr>
<tr>
<td>-11</td>
<td>Warning: The entity or list of entities had no users.</td>
</tr>
</tbody>
</table>

No list is created for failures or warnings.
### 4.4.2 Query Operations - Application Lists And Constituent Lists

Table 4-9 shows routines that query application lists and constituent lists:

**TABLE 4-9**

**QUERY OPERATIONS - APPLICATION AND CONSTITUENT LISTS**

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>ROUTINE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Determine the number of entries in a list.</td>
<td>MALNO</td>
</tr>
<tr>
<td>Find the position in a list of a specified entity key.</td>
<td>MALFND</td>
</tr>
<tr>
<td>Read the entity key at the specified position in the list.</td>
<td>MALGTK</td>
</tr>
<tr>
<td>Read the next entry in a list.</td>
<td>MALRD</td>
</tr>
<tr>
<td>Setup for reading in a forward direction.</td>
<td>MALSTF</td>
</tr>
<tr>
<td>Setup for reading in reverse direction.</td>
<td>MALSTR</td>
</tr>
</tbody>
</table>

Routines are further described on the following pages.
MALNO

FUNCTION: Count the entities in a list.

FORMAT: MALNO(KEY1,KOUNT,IRC)

INPUT:
  KEY1 : ANYKEY
  The entity or list of entities to be counted

OUTPUT:
  KOUNT : INTEGER
  The number of entities in the list.

  IRC : INTEGER
  The return code.

DESCRIPTION: KEY1 may be either an entity key or a list key. If KEY1 is an
entity the number of constituents is returned. If KEY1 is a
list the number of entities on the list is returned.

The possible return code values are:

  0 = Success
  17 = Failure  KEYS is not an entity or an
  38, 39, 40 = Failure  application list.

  KOUNT is returned zero for all failures.
**FUNCTION:**

Find the position of an entity in a list. If KEY1 is an entity, find its position in the constituent list of that entity.

**FORMAT:**

MALFND(KEY1, KEY2, IFIRST, IPOS, IRC)

**INPUT:**

- **KEY1** : ANYKEY  
  The entity or list of entities in which KEY2 is to be found.
- **KEY2** : ENTKEY  
  The entity to be found in KEY1.
- **IFIRST** : INTEGER  
  The position in KEY1 where the find operation is to start.

**OUTPUT:**

- **IPOS** : INTEGER  
  The position in KEY1 where KEY2 is found.
- **IRC** : INTEGER  
  The return code.

**DESCRIPTION:**

KEY1 may be either an entity key or a list key. If KEY1 is a list then KEY2 is found in the list. If KEY1 is an entity, then KEY2 is found in the constituent list of KEY1. KEY2 must be an entity key. The find starts at position IFIRST. Each entity in KEY1 is examined for equality with KEY2 starting with the specified position. If a match is found, then the position is returned in IPOS. If there is no match, then IPOS is returned as zero and IRC signals an error. If there are multiple matches, then only the first (leftmost) match is returned in IPOS.

The possible return code values are:

- **0** = Success
- **17** = Failure  
  KEY1 is not an entity or an application list.
- **18** = Failure  
  KEY1 is nil.
- **25** = Failure  
  No match was found.
- **38, 39, 40** = Failure  
  No more core memory.
MALGTK

FUNCTION: Get the Nth Key from a list.

FORMAT: MALGTK(KEY1, IPOS, KEY2, IRC)

INPUT:
   KEY1 : ANYKEY
   The entity or list of entities to be processed.

   IPOS : INTEGER
   The position in the list of the target entity.

OUTPUT:
   KEY2 : ENTEKEY
   The requested key.

   IRC : INTEGER
   The return code.

DESCRIPTION: If KEY1 is a list, get the IPOS entry from the list. If KEY1
is an entity, get the IPOS entry from the constituent list of KEY1.

The possible return code values are:

   0 = Success
   14 = Failure
       IPOS is outside the range of the application list.
   17 = Failure
       KEY1 is not an entity or an application list.
   38, 39, 40 = Failure
       No more core memory.
MALRD

FUNCTION: Read the next entry in a list.

FORMAT: MALRD(KEY1,KEY2,IRC)

INPUT:
KEY1 : ANYKEY
The entity or list of entities to be read.

OUTPUT:
KEY2 : ENTIKEY
The entity of the next list entry. Next depends on the direction of the read set by MALSTF or MALSTR.

IRC : INTEGER
The return code.

DESCRIPTION: The next entity in the list is returned. We recommend setting the direction by using MALSTF or MALSTR before the first time this routine is used to read a list.

The possible return code values are:

0 = Success
17 = Failure KEY1 is not a valid entity key or list key.
32 = Failure Cannot read constituent list.
33 = Failure Cannot read constituent list.
38, 39, 40 = Failure No more core memory exists.
-5 = Warning End of list reached.

4-58
FUNCTION: Setup for reading in forward direction.

FORMAT: MALSTF(KEY1,IRC)

INPUT:
KEY1 : ANYKEY
The entity or list of entities to be processed in a forward direction.

OUTPUT:
IRC : INTEGER
The return code.

DESCRIPTION: If KEY1 is an entity, then the constituent list of KEY1 will be set up for forward processing. If KEY1 is an application list, the list will be set up for forward processing.

The possible return code values are:

0 = Success
17 = Failure
    KEY1 is not a valid entity key or list key.
18 = Failure
    KEY1 is nil.
32 = Failure
    Cannot read constituent list.
33 = Failure
    Cannot read constituent list.
38, 39, 40 = Failure
    No more core memory.
MALSTR

FUNCTION: Setup for reading in reverse direction.

FORMAT: MALSTR(KEY1,IRC)

INPUT:
   KEY1 : ANYKEY
   The entity or list of entities to be processed in the reverse direction.

OUTPUT:
   IRC : INTEGER
   The return code.

DESCRIPTION: If KEY1 is an entity, then the constituent list of KEY1 will be set up for reverse processing. If KEY1 is an application list, the list will be set up for reverse processing.

The possible return code values are:

0 = Success
17 = Failure
   KEY1 is not a valid entity key or list key.
18 = Failure
   KEY1 is nil.
32 = Failure
   Cannot read constituent list.
33 = Failure
   Cannot read constituent list.
38, 39, 40 = Failure
   No more core memory.
4.4.3 Update Operations - Application Lists and Constituent Lists

Table 4-10 presents the update routines that apply to both application lists and constituent lists:

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>ROUTINE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attach an entity or list of entities to a list.</td>
<td>MALATC</td>
</tr>
<tr>
<td>Insert an entity or list of entities into a list.</td>
<td>MALINS</td>
</tr>
<tr>
<td>Remove an entity from a list.</td>
<td>MALRMV</td>
</tr>
<tr>
<td>Replace an entity in a list.</td>
<td>MALRPL</td>
</tr>
<tr>
<td>Replace a list (all of the entries).</td>
<td>MALREP</td>
</tr>
<tr>
<td>Reverse the order of a list.</td>
<td>MALRVS</td>
</tr>
</tbody>
</table>

Routines are further described on the following pages.
FUNCTION: Attach an entity or list of entities to a list. If KEY1 is an entity then attach to the constituent list of that entity.

FORMAT: MALATC(KEY1,KEY2,IRC)

INPUT:
KEY1 : ANYKEY
The entity or list of entities to which KEY2 is to be attached.

KEY2 : ANYKEY
The entity or list to be attached to KEY1.

OUTPUT:
IRC : INTEGER
The return code.

DESCRIPTION: KEY1 may be either an entity key or a list key. If KEY1 is a list, then KEY2 is attached to the list. If KEY1 is an entity, then KEY2 is attached to the constituent list of KEY1. This will make KEY2 a constituent of KEY1. KEY2 may be either an entity key or a list key. If KEY2 is a list, then the entire list is attached to KEY1. This is the same as doing multiple attaches of an entity. If KEY2 is an entity, then the entity is attached to KEY1.

EXAMPLE: See Sample Programs Section.

The possible return code values are:

0 = Success
9 = Failure KEY1 is nil.
10 = Failure KEY1 is not an entity or an application list.
38, 39, 40 = Failure No more core memory.
MALINS

FUNCTION: Insert an entity or list of entities into a list. If KEY1 is an entity, then insert into the constituent list of that entity.

FORMAT: MALINS(KEY1,KEY2,IPOS,IRC)

INPUT:
KEY1 : ANYKEY
The entity or list of entities in which KEY2 is to be inserted.

KEY2 : ANYKEY
The entity or list to be inserted in KEY1.

IPOS : INTEGER
The position in KEY1 where the insert is to take place.

OUTPUT:
IRC : INTEGER
The return code.

DESCRIPTION: KEY1 may be either an entity key or a list key. If KEY1 is a list, then KEY2 is inserted in the list. If KEY1 is an entity, then KEY2 is inserted in the constituent list of KEY1. KEY2 may be either an entity key or a list key. If KEY2 is a list, then the entire list is inserted in KEY1. If KEY2 is an entity, then the entity is inserted in KEY1.

The insert takes place before IPOS. That is, the entity at IPOS is moved by one position if KEY2 is an entity or by the number of elements in the list if KEY2 is a list. Then the elements are moved into the vacated positions.

The possible return code values are:

0 = Success
14 = Failure
17 = Failure
18 = Failure
32 = Failure
33 = Failure
38, 39, 40 = Failure

Given position is outside range of list.
KEY1 is not an entity or an application list.
KEY1 is nil.
Cannot read constituent list.
Cannot read constituent list.
No more core memory.
FUNCTION: Remove an entity from a list. If KEY1 is an entity, then remove it from the constituent list of that entity.

FORMAT: MALRMV(KEY1,IPOS,IRC)

INPUT:
- KEY1: ANYKEY
  The entity or list of entities from which an entity is to be removed.
- IPOS: INTEGER
  The position, in the list, of the entity to be removed.

OUTPUT:
- IRC: INTEGER
  The return code.

DESCRIPTION: KEY1 may be either an entity key or a list key. If KEY1 is a list, then an entity is removed from the list. If KEY1 is an entity, then an entity is removed from the constituent list of KEY1. IPOS is the position number of the entity to be removed. The MAS delete rules are used to see if the entity can be removed from the constituent list. If the entity can be removed, it is removed; and if it is the last constituent in the list, the list is deleted. If the entity removed is marked for delete, an attempt to delete the entity will occur.

The possible return code values are:

0 = Success
14 = Failure
17 = Failure
18 = Failure
27 = Failure
32 = Failure
33 = Failure
38, 39, 40 = Failure
42 = Failure

IPOS is outside range of list.
KEY1 is not a valid entity key or list key.
KEY1 is nil.
Delete rules prohibit delete.
Cannot read constituent list.
Cannot read constituent list.
No more core memory.
Delete rules defined incorrectly.
FUNCTION: Replace an entity in a list. If KEY1 is an entity, then replace in the constituent list of that entity.

FORMAT: MALRPL(KEY1, KEY2, IPOS, IRC)

INPUT:
  KEY1 : ANYKEY
  The entity or list of entities in which an entity is to be replaced.
  KEY2 : ENTKEY
  The entity that will replace an entity in KEY1.
  IPOS : INTEGER
  The position of the entity in KEY1 to be replaced.

OUTPUT:
  IRC : INTEGER
  The return code.

DESCRIPTION: KEY1 may be either an entity key or a list key. If KEY1 is a list, then an entity is replaced in the list. If KEY1 is an entity, then an entity is replaced in the constituent list of KEY1. KEY2 must be an entity key. The entity at position IPOS in KEY1 will be replaced by KEY2. If the entity being replaced is "MARKED FOR DELETE," then an attempt is made to delete the entity.

The possible return code values are:

0 = Success
14 = Failure  IPOS is outside range of list.
17 = Failure  KEY1 is not a valid entity key or list key.
18 = Failure  KEY1 is nil.
27 = Failure  Delete rules prohibit delete.
32 = Failure  Cannot read constituent list.
33 = Failure  Cannot read constituent list.
38, 39, 40 = Failure  No more core memory exists.
MALREP

FUNCTION: Replace a list. If KEY1 is an entity, then replace the constituent list of that entity.

FORMAT: MALREP(KEY1,KEY2,IRC)

INPUT:
   KEY1 : ANYKEY
   The entity or list of entities to be replaced.
   KEY2 : ANYKEY
   The entity or list to replace KEY1.

OUTPUT:
   IRC : INTEGER
   The return code.

DESCRIPTION: KEY1 may be either an entity key or a list key. If KEY1 is a list, then KEY2 replaces KEY1. If KEY1 is an entity, then the constituent list of KEY1 is replaced by KEY2. KEY2 may be either an entity or a list key.

The possible return code values are:

   0 = Success
   17 = Failure KEY1 is not a valid entity key or list key.
   18 = Failure KEY1 is nil.
   32 = Failure Cannot read constituent list.
   33 = Failure Cannot read constituent list.
   38, 39, 40 = Failure No more core memory.
MALRVS

FUNCTION: Reverse the order of the entities in a list.

FORMAT: MALRVS(KEY1,IRC)

INPUT:
  KEY1 : ANYKEY
         The entity or list of entities in which the order of the entities is to be reversed.

OUTPUT:
  IRC  : INTEGER
         The return code.

DESCRIPTION: KEY1 may be either an entity key or a list key. If KEY1 is a list, then the list is reversed. If KEY1 is an entity, then the constituent list is reversed.

The possible return code values are:

  0 = Success
  17 = Failure
       KEY1 is not a valid entity key or list key.
  38, 39, 40 = Failure
                No more core memory exists.
  -6 = Warning
       KEY1 is empty.
4.4.4 Update Operations – Application Lists Only

Table 4-11 shows routines that update an application list (they do not apply to constituent lists):

**TABLE 4-11**

**UPDATE OPERATIONS – APPLICATION LISTS**

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>ROUTINE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reset an application list to be reused.</td>
<td>MALRST</td>
</tr>
<tr>
<td>Remove duplicate entries from an application list.</td>
<td>MALRDE</td>
</tr>
<tr>
<td>Sort the entries in an application list into user-constituent order.</td>
<td>MALROR</td>
</tr>
<tr>
<td>Sort the entries in an application list into inclusive user-constituent order.</td>
<td>MALRRI</td>
</tr>
</tbody>
</table>

Routines are further defined in the following pages.
MALKST

FUNCTION:  Reset an application list.

FORMAT:   MALKST(KEYL,RC)

INPUT:    
    KEYL : LISTKEY
        The key of the application list whose entries
        are to be reset.

OUTPUT:   
    RC : INTEGER
        The return code.

DESCRIPTION: Given an application list that has one or more entries, all
entries will be removed from the list, thus maintaining the
size of the original list.

The possible return code values are:

    0 = Success
    17 = Failure  KEYL is not a valid entity key
                 or list key.
    18 = Failure  KEYL is nil.
    38, 39, 40 = Failure  No more core memory.
MALRDE

FUNCTION: Remove duplicate entries in a list.

FORMAT: MALRDE(KEY1,IRC)

INPUT:
KEY1 : LISTKEY
The input/output list.

OUTPUT:
IRC : INTEGER
The return code.

DESCRIPTION: Any duplicate entities found in the input list will be removed. The change is made in-place. The first instance of each entity will be kept.

The possible return code values are:

0 = Success
17 = Failure KEY1 is not a valid entity key or list key.
26 = Failure Duplicates not removed.
38, 39, 40 = Failure No more core memory.
MALAROR

FUNCTION: Reorder a list of entities so that the users appear at the head of the list.

FORMAT: MALAROR(KEYL, IRC)

INPUT:
  KEYL : LISTKEY
  Key of an application list.

OUTPUT:
  IRC : INTEGER
  Return code
  0 = Good return
  <0 Critical error
  >0 Warning

DESCRIPTION: For each member of the list, search each of the remaining members for its users; put users at the head of the list.

The possible return code values are:

  0 = Success
  15 = Error
      A list has too many members.
  17 = Error
      Input key not a list key.
  18 = Error
      Input key is nil.
  38 = Error
      No more core available.
MALARRI (MALARORI)

FUNCTION: Sort the entities in an application list in inclusive user to constituent order.

FORMAT: MALRRI(KEY,RRC) or MALARORI(KEY,RRC)

INPUT:
   KEY : ANYKEY
   The key to an entity or list of entities.

OUTPUT:
   RRC : INTEGER
   The return code.

DESCRIPTION: For each entity on the input list, its user list is expanded inclusively. All users appearing on the input list will occur before their constituents.

The possible return code values are:

   0 = Success
   17 = Failure
       KEY1 is not a valid entity key or list key.
   18 = Failure
       KEY1 is nil.
   38, 39, 40 = Failure
       No more core memory.
4.4.5 Boolean Operations - Application Lists and Constituent Lists

For Boolean operations, there are two input lists and one output list. The application is responsible for providing two input lists consistent with the Boolean operation to be performed. No validation checking is done. If one or both of the input lists contain duplicate entities, then the output list may also contain duplicate entities. This result may not be consistent with the Boolean theory operation being performed.

The Boolean routines are shown in Table 4-12, and the following pages.

**TABLE 4-12**

**BOOLEAN ROUTINES**

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>ROUTINE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create a list from a Boolean &quot;AND&quot; on two input lists.</td>
<td>MALAND</td>
</tr>
<tr>
<td>Create a list from a Boolean &quot;NOT&quot; on two input lists.</td>
<td>MALNOT</td>
</tr>
<tr>
<td>Create a list from a Boolean &quot;OR&quot; on two input lists.</td>
<td>MALOR</td>
</tr>
</tbody>
</table>
FUNCTION: Create a list from a Boolean "AND" on two input lists.

FORMAT: MALAND(KEY1,KEY2,KEY3,IRC)

INPUT:
KEY1 : ANYKEY
An entity or a list that is to be AND'ed.
KEY2 : ANYKEY
An entity or a list that is to be AND'ed.

OUTPUT:
KEY3 : LISTKEY
The list of entities that occurred in both KEY1 and KEY2.
IRC : INTEGER
The return code.

DESCRIPTION: KEY3 is initialized to nil. KEY1 may be either an entity key or a list key. If KEY1 is an entity key, then the constituent list of KEY1 is AND'ed with KEY2. If KEY1 is a list key, then KEY1 is AND'ed with KEY2. KEY2 may be either an entity key or a list key. If KEY2 is an entity key then the constituent list of KEY2 is AND'ed with KEY1. If KEY2 is a list key then KEY2 is AND'ed with KEY2. The list KEY3 is created, corresponding to the set theoretical intersection of KEY1 and KEY2.

The possible return code values are:

- 0 = Success
- 17 = Failure KEY1 is not a valid entity key or list key.
- 18 = Failure KEY1 is nil.
- 38, 39, 40 = Failure No more core memory.
- -6 = Warning There were no entities in common.

No list is created for failures or warnings.
MALNOT

FUNCTION: Create a list from a Boolean "NOT" on two input lists.

FORMAT: MALNOT(KEY1,KEY2,KEY3,IRC)

INPUT:
   KEY1 : ANYKEY
      An entity or a list that is to be NOT'ed.
   KEY2 : ANYKEY
      An entity or a list that is to be NOT'ed.

OUTPUT:
   KEY3 : LISTKEY
      The list of entities that occurred in KEY1 but not in KEY2.
   IRC : INTEGER
      The return code.

DESCRIPTION: KEY3 is initialized to nil. KEY1 may be either an entity key or a list key. If KEY1 is an entity key, then the constituent list of KEY1 is NOT'ed with KEY2. If KEY1 is a list key, then KEY1 is NOT'ed with KEY2. KEY2 may be either an entity key or a list key. If KEY2 is an entity key, then the constituent list of KEY2 is NOT'ed with KEY1. If KEY2 is a list key, then KEY2 is NOT'ed with KEY1. The list KEY3 is created, corresponding to the set theoretical difference of KEY1 and KEY2.

The possible return code values are:

   0 = Success
   17 = Failure
      KEY1 is not a valid entity key or list key.
   38, 39, 40 = Failure
      No more core memory.
   -6 = Warning
      There was no difference between the two lists.

No list is created for failures or warnings.
MALOR

FUNCTION: Create a list from a Boolean "OR" on two input lists.

FORMAT: MALOR(KEY1,KEY2,KEY3,IRC)

INPUT:
  KEY1 : ANYKEY
  An entity or a list that is to be OR'ed.

  KEY2 : ANYKEY
  An entity or a list that is to be OR'ed.

OUTPUT:
  KEY3 : LISTKEY
  The list of entities that occurred in either KEY1 or KEY2.

  IRC : INTEGER
  The return code.

DESCRIPTION: KEY1 may be either an entity key or a list key. If KEY1 is an entity key, then the constituent list of KEY1 is OR'ed with KEY2. If KEY1 is a list key, then KEY1 is OR'ed with KEY2. KEY2 may be either an entity key or a list key. If KEY2 is an entity key, then the constituent list of KEY2 is OR'ed with KEY1. If KEY2 is a list key, then KEY2 is OR'ed with KEY1. The list KEY3 is created, corresponding to the set theoretical union of KEY1 and KEY2. There will be no duplicates in KEY3.

The possible return code values are:

  0 = Success
  17 = Failure KEY1 is not a valid entity key or list key.
  18 = Failure KEY1 is nil.
  38, 39, 40 = Failure No more core memory.
  -6 = Warning Neither key has constituents nor application list.

No list is created for failures or warnings.
4.4.6 Delete Operations – Application Lists Only

Table 4-13 presents the delete routines:

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>ROUTINE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delete an application list.</td>
<td>MALD</td>
</tr>
<tr>
<td>Delete all application lists.</td>
<td>MALDA</td>
</tr>
<tr>
<td>Delete an application list and all lists created after it.</td>
<td>MALDI</td>
</tr>
<tr>
<td>Set or unset the application list lock flag.</td>
<td>MALOCK</td>
</tr>
</tbody>
</table>

Delete routines are further described on the following pages.
FUNCTION: Delete an application list.

FORMAT: MALD(KEY1,IRC)

INPUT:
   KEY1 : LISTKEY
   The list to be deleted.

OUTPUT:
   IRC : INTEGER
   The return code.

DESCRIPTION: KEY1 must not be an entity key. KEY1 is deleted. KEY1 cannot be recovered.

The possible return code values are:

0 = Success
29 = Failure KEY1 is not a list key.
38, 39, 40 = Failure No more core memory exists.
FUNCTION: Delete all application lists.

FORMAT: MALDA(IRC)

INPUT:
NONE

OUTPUT:
IRC : INTEGER
The return code.

DESCRIPTION: All application lists will be deleted. They cannot be recovered. If an application list is locked, then it will not be deleted.

The possible return code values are:

38, 39, 40 = Failure
No more core memory exists.
MALDI

FUNCTION: Delete an application list and all lists created after it.

FORMAT: MALDI(KEY1,IRC)

INPUT:
KEY1 : LISTKEY
The list to be deleted.

OUTPUT:
IRC : INTEGER
The return code.

DESCRIPTION: KEY1 must not be an entity key. The list identified by KEY1 and all lists created after it will be deleted. Deleted lists cannot be recovered. If an application list is locked, then it will not be deleted.

The possible return code values are:

17 = Failure
KEY1 is not a list or no lists exist.

38, 39, 40 = Failure
No more core memory exists.
MALOCK

FUNCTION:  Set or unset the application list lock flag.

FORMAT:   MALOCK(KEY1,LOCK,IRC)

INPUT:
   KEY1  :  LISTKEY
   The list to be set.

   LOCK  :  INTEGER
   The lock setting
   =0  unlocked
   =1  locked

OUTPUT:
   IRC   :  INTEGER
   The return code

DESCRIPTION: A list that is locked will not be deleted by the MAS interface
functions MALDA or MALDI. All other functions that delete
lists will delete a locked list.

The possible return code values are:

   17 = Failure    KEYL is not a list.
   38, 39, 40 = Failure    No more core memory exists.
4.5 **Execute Operations**

Many times in an application program, it is necessary to process each entity on a list with a user-written subroutine. An example of how each entity of a particular KIND could be processed is shown in the following FORTRAN example:

```
CALL MALK(KNDVAL, LIST, IRC)
CALL MALSTF(LIST, IRC)
100 CALL MALRD(LIST, ENTITY, IRC)
   IF(IRC .NE. 0) GOTO 200
   CALL MAEGTK(ENTITY, ADB, IRC)
   CALL USRSUB(ENTITY, ADB, USRDAT, URC)
   GOTO 100
200 CALL MALD(LIST, IRC)
```

Similar techniques could be used for processing an application list, a constituent list, or a user list. Note that the return code from the user-written subroutine may affect whether processing should continue.

The **EXECUTE** operations, however, provide a simpler, more efficient method. The previous example can be replaced with:

```
CALL MAKXEQ(KNDVAL, USRDAT, USRSUB, URC, IRC)
```
An EXECUTE operation, shown in Figure 4-2, is invoked by a user routine to execute a user-written subroutine for each entity on a list.

![Figure 4-2. Execute Operation](image)

The user-written subroutine is passed as a parameter from the user routine to the EXECUTE operation. The method by which this is done in the multi-language environment supported by MAS is described in Section 3, which describes both the IBM and VAX operating environments.

The user-written subroutine must use a standard parameter list. The following example shows the data types for the standard parameter list:

```
USRSUB(KEY, ADB, USRDAT, URC)
```

where the data type for

- **KEY** is an ENTKEY (input).
- **ADB** is an ENTBLOCK (input/output; note that the KIND, LENGTH, and SYSUSE fields should not be changed).
- **USRDAT** is user-defined (input/output).
- **URC** is an INTEGER (output).
The possible return code values from the user-written subroutine affect the EXECUTE operations as follows:

0 through 7  The EXECUTE operation will continue processing.

< 0 or > 7   The EXECUTE operation will halt processing and return the URC to the user routine that called the EXECUTE operation.

There are specific meanings within the above ranges for some of the EXECUTE operations. These are explained in the routine descriptions.

Table 4-14 shows the EXECUTE routines:

**TABLE 4-14**

<table>
<thead>
<tr>
<th>EXECUTE ROUTINES</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESCRIPTION</td>
</tr>
<tr>
<td>Execute a procedure on an entity or a list of entities.</td>
</tr>
<tr>
<td>Execute a procedure on all entities of a specified KIND.</td>
</tr>
<tr>
<td>Execute a procedure on an entity or a list of entities.</td>
</tr>
<tr>
<td>Execute a procedure on the constituents of an entity.</td>
</tr>
<tr>
<td>Execute a procedure on the users of an entity.</td>
</tr>
<tr>
<td>Execute a sorting procedure on the constituents of an entity or the entities in a list.</td>
</tr>
</tbody>
</table>

Routines are described in the following pages.
MAEXEQ

FUNCTION: Execute a procedure on a entity or a list of entities.

FORMAT: MAEXEQ(KEY1,DATA,PROC,RCC,IRC)

INPUT:
  KEY1: ANYKEY
  The entity or list of entities on which the application procedure should be performed.

  DATA: BLKDATA
  The application-defined data structure that is passed to the application-defined procedure.

  PROC: ROUTINE
  Application-defined procedure that processes one entity at a time.

OUTPUT:
  RCC: INTEGER
  The procedure PROC return code.

  IRC: INTEGER
  The HAS return code.

DESCRIPTION: The entity, or each entity on an application list, is passed to the application-defined procedure. The operation performed on the entity is determined by the application-defined procedure. Processing of application lists occurs in the forward direction. Unless the default direction has been changed by previous processing or by a call to MALSTF or MALSTR. Processing halts when the application-defined procedure returns RCC > 7 or RCC < 0 after processing an entity.

The possible return code values are:

- 0 = Success
- 17 = Failure
- 18 = Failure
- 23 = Execution Halted
- 24 = Execution Halted
- 38, 39, 40 = Failure
- -5 = Warning

KEY1 is not a valid entity key or list key.
KEY1 is nil.
RCC > 7 and RCC <= 15.
RCC > 15 or RCC < 0.
No more core memory exists.
End of list reached.
FUNCTION: Execute a procedure on all entities of a specified kind.

FORMAT: MAKXEQ(KIND,DATAPROC,RCC,IRC)

INPUT:
- KIND: INTEGER
  The KIND value of the entities to be processed.
- DATA: BLKDATA
  The application-defined data structure, which is passed to the application-defined procedure.
- PROC: ROUTINE
  Application-defined procedure that processes one entity at a time.

OUTPUT:
- RCC: INTEGER
  The procedure PROC return code.
- IRC: INTEGER
  The MAS return code.

DESCRIPTION: Each entity of the specified kind is passed to the application-defined procedure. The order of processing is in the reverse order for which the entities were created. When the application-defined procedure returns $RRC \geq 7$ or $RRC < 0$, processing halts.

The possible return code values are:

- 0 = Success
- 23 = Execution Halted
- 24 = Execution Halted
- 38, 39, 40 = Failure
- -1 = Warning

$RRC > 7$ and $RRC \leq 15$.
$RRC > 15$ or $RRC < 0$.
No more core memory exists.
No such kind exists.
MALXEQ

FUNCTION: Execute a procedure on an entity or a list of entities. Construct an output list of entities as determined by the application procedure.

FORMAT: MALXEQ(KEY1, DATA, PROC, KEY2, RCC, IRC)

INPUT:
KEY1 : ANYKEY
The entity or list of entities to be processed.

DATA : BLKDATA
The application-defined data structure, which is passed to the application-defined procedure.

PROC : ROUTINE
Application-defined procedure that processes one entity at a time.

OUTPUT:
KEY2 : LISTKEY
The list created by this function.

RCC : INTEGER
The procedure PROC return code.

IRC : INTEGER
The return code produced by this operation.

DESCRIPTION: An empty list (KEY2) is created. The entity, or each entity in sequence if a list is supplied, is passed to the application-defined procedure. The operation performed on the entity is determined by the application-defined procedure. When the application-defined procedure return code is "success," (RCC = 0 or 1), the entity just processed is added to the result list. When an application-defined procedure returns code is "failure", (<0 or > 7), MALXEQ is terminated. When an application-defined procedure return code is "warning" (2 through 7), the entity just processed is not placed on the result list, but processing continues.

The processing of application lists occurs in the forward direction unless the default direction has been changed by previous processing or by a call to MALSTF or MALSTR.
The possible return code values are:

0 = Success
17 = Failure

18 = Failure
KEY1 is not a valid entity key or list key.

23 = Execution Halted
RRC > 7 and RRC ≤ 15.

24 = Execution Halted
RRC > 15 or RRC < 0.

32 = Failure
Cannot read constituent list.

33 = Failure
Cannot read constituent list.

38, 39, 40 = Failure
No more core memory exists.

-2 = Warning
An entity processed with RRC > 1 and RRC ≤ 7.

-6 = Warning
No entities processed with RRC = 0 or RRC = 1.
FUNCTION: Given an application-defined procedure, perform this procedure on the constituents of an entity or list of entities.

FORMAT: MAECXQ(KEY1,DATAREC,PROCNM,KEY2,RRC,IRC)

INPUT:

KEY1 : ANYKEY
Key of an entity or an application list that is constituent(s) are to be processed.

DATAREC : BLKDATA
The application-defined data structure that is passed to the application-defined procedure.

PROCNM : ROUTINE
Application-defined procedure that processes one entity at a time.

OUTPUT:

KEY2 : LISTKEY
Key to the list of constituents that processed with RRC = 0, 1; routine will append to KEY2 if a valid list key.

RRC : INTEGER
Return code of the application-defined procedure.

IRC : INTEGER
Return code

DESCRIPTION: Each constituent of an entity is processed by the application-defined procedure. Each constituent processed with RRC = 0 or 1 is added to KEY2. For an application list, entities are processed in the direction set by previous processing or by a call to MALSTF or MALSTR; the constituents of these entities are processed in the forward direction only. (Each constituent processed with RRC = 0 or 1 is added to KEY2.) Processing halts when the application-defined procedure returns RRC > 7 or RRC < 0 after processing a constituent.
The possible return code values are:

0 = Success
17 = Failure
18 = Failure
23 = Execution Halted
24 = Execution Halted
32 = Failure
33 = Failure
38, 39, 40 = Failure
-2 = Warning
-6 = Warning

KEY1 is not a valid entity key or list key.
KEY1 is nil.
RRC > 7 and RRC ≤ 15.
RRC > 15 or RRC < 0.
Cannot read constituent list.
Cannot read constituent list.
No more core memory exists.
An entity processed with RRC > 1 and RRC ≤ 7.
No entities processed with RRC = 0 or RRC = 1.
MAEUXO

FUNCTION: Given an application-defined procedure, perform this procedure on the users of an entity or list of entities.

FORMAT: MAEUXQ(KEY1,DATAREC,PROCNM,KEY2,RRC,IRC)

INPUT:

KEY1 : ANYKEY
Key of an entity or an application list that is user(s) are to be processed.

DATAREC : BLKDATA
The application-defined data structure that passed to the application-defined procedure.

PROCNM : ROUTINE
Application-defined procedure that processes one entity at a time.

OUTPUT:

KEY2 : LISTKEY
Key to the list of users that processed with RRC = 0, 1; routine will append if given a valid list key.

RRC : INTEGER
Return code of the application-defined procedure.

IRC : INTEGER
Return code

DESCRIPTION: Each user of an entity is processed by the application-defined procedure. Each user processed with RRC = 0 or 1 is added to KEY2. For an application list, entities are processed in the direction set up by previous processing or by a call to MALSTF or MALSTR; the users of these entities are processed in the forward direction only. (Each user processed with RRC = 0 or 1 is added to KEY2. Processing halts when the application-defined procedure returns RRC > 7 or RRC < 0 after processing a user.

4-91
The possible return code values are:

0 = Success
17 = Failure
18 = Failure
23 = Execution Halted
24 = Execution Halted
32 = Failure
33 = Failure
38, 39, 40 = Failure
-2 = Warning
-6 = Warning

KEY1 is not a valid entity key or list key.
KEY1 is nil.
RRC > 7 and RRC < 15.
RRC > 15 or RRC < 0.
Cannot read constituent list.
Cannot read constituent list.
No more core memory exists.
An entity processed with RRC > 1 and RRC < 7.
No entities processed with RRC = 0 or RRC = 1.
FUNCTION: MALSRT

Sort the constituents of an entity or the entities in an application list according to an application-defined algorithm.

FORMAT: MALSRT(KEY,PROCNAME,RRC)

INPUT:

KEY : ANYKEY
The key to an entity or list of entities.

PROCNAME : ROUTINE
ROUTINE(CURRENT:ENTBLOCK;
NEXT :ENTBLOCK;
FLIP :BOOLEAN;
RR :IRC);

Application-supplied routine with the above parameter list that determines the relative value of two entities in the input list.

OUTPUT:

RRC : INTEGER
The return code.

DESCRIPTION: Given an entity or list of entities using a combination of bubblesort and quicksort, MALSRT sends a user-defined routine two entity ADBs. The routine returns a Boolean value indicating true if the first ADB is greater than the second or the second is less than or equal to the first entity.

The possible return code values are:

0 = Success
17 = Failure KEY1 is not a valid entity key or list key.
18 = Failure KEY1 is nil.
23 = Failure RR <> 0.
38, 39, 40 = Failure No more core memory exists.
### 4.6 General Purpose Utilities

This section contains descriptions of general purpose utility routines, as shown in Table 4-15, and the following pages.

#### TABLE 4-15
GENERAL PURPOSE UTILITIES

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>ROUTINE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get number of different KIND values in the working-form model.</td>
<td>MAECTK</td>
</tr>
<tr>
<td>Get KIND value stored at specific position in KIND list.</td>
<td>MAEKND</td>
</tr>
<tr>
<td>Get actual model space used and amount of model free space.</td>
<td>MASMSZ</td>
</tr>
<tr>
<td>Determine the number of entities in the model of a specified KIND.</td>
<td>MAKCNT</td>
</tr>
</tbody>
</table>
MAECTK

FUNCTION: Get the number of different KIND values in the working-form model.

FORMAT: MAECTK(KNDCNT,IRC)

INPUT: NONE

OUTPUT:
   KNDCNT : INTEGER
            The number of different KIND values in the working-form model.

   IRC : INTEGER
         The return code.

DESCRIPTION: Get the number of KIND values in the working-form model from the KIND list.

NOTE: Works in conjunction with MAEKND.

The possible return code values are:

   0 = Success
   30 = Failure
   38, 39, 40 = Failure

   No model established.
   No more core memory; had no constituents.
MAEKND

FUNCTION: Get KIND value at specified position in the KIND list.

FORMAT: MAEKND(KNDPOS,KNDVAL,IRC)

INPUT:
   KNDPOS : INTEGER
     The position in the standard array of where to get the KIND value.

OUTPUT:
   KNDVAL : INTEGER
     The KIND value retrieved from the KIND list.

   IRC : INTEGER
     The return code.

DESCRIPTION: Get the KIND value at KNDPOS in the KIND list.

NOTE: Works in conjunction with MAECTK.

The possible return code values are:

   0 = Success
   30 = Failure
     No model established.
   31 = Failure
     Corrupted model.
   38, 39, 40 = Failure
     No more core memory exists.
FUNCTION: Determine actual model used space and model free space (in bytes).

FORMAT: MASMSZ(MODSIZ,FRESIZ,IRC)

OUTPUT:

MODSIZ : INTEGER
The total number of bytes used by the model.

FRESIZ : INTEGER
The total number of bytes of free space.

IRC : INTEGER
The return code.

DESCRIPTION: The used model space is calculated by taking the difference of allocated model space and free model space. This routine can only be used where the MAS memory manager is used.

The possible return code values are:

0 = Success
MAKCNT

FUNCTION: Determine the number of entities in the model of a specified KIND.

FORMAT: MAKCNT(KIND,COUNT,IRC)

INPUT:
KIND : INTEGER
The KIND value for which a count is to be determined.

OUTPUT:
COUNT : INTEGER
The number of entities in the model of the specified KIND.
IRC : INTEGER
The return code.

DESCRIPTION: If the KIND specified is in the model, determine the number of entities with that KIND.

The possible return code values are:
0 = Success
4.7 Special Purpose Utilities

This section contains descriptions of special purpose utilities, as shown in Table 4-16, and the following pages.

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>ROUTINE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delete an entity or list of entities but do not consider delete rules.</td>
<td>MIDBD</td>
</tr>
<tr>
<td>Remove an entity from the constituent list and delete if marked.</td>
<td>MIDBRE</td>
</tr>
<tr>
<td>Retrieve the run-time subschema for a given entity kind.</td>
<td>MARSGT</td>
</tr>
<tr>
<td>Create a run-time subschema for a given entity kind.</td>
<td>MRSCR</td>
</tr>
<tr>
<td>Delete the run-time subschema for a given entity kind.</td>
<td>MARDLT</td>
</tr>
</tbody>
</table>
MIDBD

FUNCTION: Delete an entity or list of entities but do not consider the delete rules.

FORMAT: MIDBD(KEY1,RC)

INPUT:
KEY1 : ANYKEY
An entity or list of entities that are to be deleted.

OUTPUT:
RC : INTEGER
The return code.

DESCRIPTION: If KEY1 is an entity, delete the entity. If KEY1 is a list, then eliminate duplicate entries and delete each entity on the list.

The possible return code values are:

0 = Success
26 = Failure
29 = Failure
32 = Failure
33 = Failure
38, 39, 40 = Failure
-6 = Warning
-11 = Warning

Duplicates not removed.
KEY1 is nil or not an entity key or a key to an application list.
Cannot read constituent list.
Cannot read constituent list.
No more core memory.
The entity or list of entities had no constituents.
KEY1 had no entities to delete.
MIDBRV

FUNCTION: Remove an entity from the constituent list or remove an entity from a list of entities and delete if marked for delete.

FORMAT: MIDBRV(KEY1,IPOS,RC)

INPUT:
KEY1 : ANYKEY
An entity or list of entities from which an entity will be removed.

IPOS : The position of the key that will be removed.

OUTPUT:
RC : INTEGER
The return code.

DESCRIPTION: If KEY1 is an entity, select the entity at the input position from the constituent list of KEY1. Remove all occurrences of it from the user list of all its users. Remove all occurrences of it from the constituent list of KEY1. If the entity is marked for delete, attempt to delete it.

If KEY1 is a list, delete it from the position indicated. If there was only one member in the list, the list is deleted.

The possible return code values are:

0 = Success
17 = Failure KEY1 is neither an entity or list key.
18 = Failure KEY1 is nil
32 = Failure Cannot read constituent list. List key.
33 = Failure Cannot read constituent list. List key.
42 = Failure Delete rules defined incorrectly.
38, 39, 40 = Failure No more core memory exists.
MARSCT

FUNCTION: To retrieve the run-time subschema for a given entity kind.

FORMAT: MARSCT(KIND,SCHPTR,RC)

INPUT:
  KIND : ORD_KIND
  The kind of the entity whose run-time subschema is to be retrieved.
  SCHPTR : T_SCHEMA_POINTER
  Pointer to the run-time subschema.

OUTPUT:
  RC : INTEGER
  The return code.

DESCRIPTION: Given a valid entity kind, a run-time subschema is located in the working form.

The possible return code values are:

37 = Failure
  A run-time subschema has not been established for this entity kind.

38, 39, 40 = Failure
  No more core memory exists.

-1 = Warning
  No such kind exists.
FUNCTION: Create a run-time subschema for a given entity kind.

FORMAT: MRSRC(KIND,SCH_SIZE,RTSS,RC)

INPUT:
  KIND : ORD_KIND
    The kind of the entity whose run-time subschema is to be added.

  SCH_SIZE : INTEGER
    The size of the schema to be created.

  RTSS : T_SCHEMA_POINTER
    The pointer to the area containing the run-time subschema.

OUTPUT:
  RC : INTEGER
    The return code.

DESCRIPTION: Given a valid entity kind, a run-time subschema is added to the working form.

The possible return code values are:

  19 = Failure
    A run-time subschema has already been established for this entity kind.

  38, 39, 40 = Failure
    No more core memory exists.

  -1 = Warning
    No such kind exists.

No subschema is created for all failures or warnings.
MARDLT

FUNCTION: To delete the run-time subschema for a given entity kind.

FORMAT: MARDLT(KIND, RC)

INPUT:
KIND : ORD_KIND
The kind of the entity whose run-time subschema is to be deleted.

OUTPUT:
RC : INTEGER
The return code.

DESCRIPTION: Given a valid entity kind, a run-time subschema is deleted from the working form.

The possible return code values are:

37 = Failure
A run-time subschema has not been established for this entity kind.

38, 39, 40 = Failure
No more core memory exists.

-1 = Warning
No such kind exists.
4.8 IBM/MVS Environment

4.8.1 Compiling Considerations

The MAS may be used by any application with the appropriate constants, types, and interface routine declarations. For PASCAL programs, these are defined in the following INCLD file.

DSNAME = CAD5.GMAP.V33.MASINC

Member = APLTPY

**INCLUDE File**

The types and constants used for the Model Access Software which are contained in the INCLUDE file member APLTPY are listed below:

```plaintext
CONST
  NULL_KEY = 0;

TYPE
  ANYKEY = INTEGER;
  EXT_RET_CODE = INTEGER;
  LISTINDEX = INTEGER;
  LISTPSTN = INTEGER;
  LISTSIZE = INTEGER;
  NAMTYP = PACKED ARRAY(.1..6.) OF CHAR;
  ORD_KIND = INTEGER;
  ROUTINE = ARRAY(.1..8.) OF CHAR;

(*)
  ENTKEY = ANYKEY;
  LISTKEY = ANYKEY;
```

The INCLUDE file also contains the formal declarations for the interface routines. The member names in the INCLUDE file are the same as the interface routine names.

4.8.2 Considerations When Using The XEO Routines (MAEXEO, MALXEO, MAKXEO, MAEXO, MAEXO, MALSRT)

If an EXECUTE routine is used, then the conventions presented in Figure 4-2 apply.

The user-written subroutine is passed as a parameter from the user routine to the EXECUTE operation. The method by which this is done in the multi-language environment supported by MAS is described below.
The name of the user subroutine is defined to the user routine in PASCAL:

```pascal
REF USRSUB : ROUTINE;
```

and in FORTRAN:

```fortran
 COMMON/USRSUB/USRSUB/
```

This allows the procedure to be passed as a parameter in a manner that is independent of the requirements of a particular language compiler. The EXECUTE operations correctly resolve this nonstandard linkage convention. A PASCAL user routine should have knowledge of the user subroutine only as a REF and not as a PROCEDURE. A PASCAL user subroutine must be declared as a SUBPROGRAM. The BLKDATA type must appear before the INCLUDE command for the formal declaration for the EXECUTE routine.

4.8.3 Linking Considerations

The MAS consists of PASCAL procedures declared as SUBPROGRAMS that have been processed by the linkage editor into a single module. The references to the PASCAL run-time support are unresolved. The module may be incorporated into an application program by the appropriate data definition statement and linkage editor control statements containing the following:

```
ddname = MASLIB
disp = SHR
dname = CAD2.GMAP.V33.LOAD

INCLUDE MASLIB(MAS)
```

The TEST library has PASCAL CHECKING enabled and contains aids for error diagnosis. The PROD library has PASCAL NOCHECK and contains minimal error diagnosis. We recommend using the TEST version during software development.

4.9 VAX/VMS Environment

4.9.1 Compiling Considerations

The MAS may be used by application with the appropriate constants, types, and interface routine declarations. For PASCAL programs, these are defined in the following INC file:

```
directory = [GMAP.V33.MASINC]
file name = APLTYP.INC
```
INCLUDE File

The types and constants used for the Model Access Software which are contained in the INCLUDE file APLTYP are listed below:

```plaintext
CONST
    NULL_KEY = 0;

TYPE
    ANYKEY = INTEGER;
    EXT_RET_CODE = INTEGER;
    LISTINDX = INTEGER;
    LISTPSTN = INTEGER;
    LISTSIZE = INTEGER;
    NAMTYP = PACKED ARRAY(.1..6.) OF CHAR;
    ORD_KIND = INTEGER;
    ROUTINE = ARRAY(.1..8.) OF CHAR;

(*)
    ENTKEY = ANYKEY;
    LISTKEY = ANYKEY;
```

The INC files also contain the formal declarations for the interface routines. The file names for the INC files are the same as the interface routine names.

4.9.2 Considerations When Using The XEO Routines (MAEXEO, MALXEO, MAKXEO, MAECXO, MAEUXO, MALSRT)

If an EXECUTE routine is used, then the conventions presented in Figure 4-2 apply.

The user-written subroutine is passed as a parameter from the user routine to the EXECUTE operation. The method by which this is done in the multi-language environment supported by MAS is described below.

The name of the user subroutine is defined to the user routine in PASCAL:

```plaintext
VAR USRSUB : [EXTERNAL]ROUTINE;
```

and in FORTRAN:

```plaintext
COMMON/USRSUB/USRSUB/
```

This allows the procedure to be passed as a parameter in a manner that is independent of the requirements of a particular language compiler. The EXECUTE operations correctly resolve this nonstandard linkage convention. A PASCAL user routine should have knowledge of the user subroutine only as an EXTERNAL VAR and not as a PROCEDURE. The BLKDATA type must appear before the INCLUDE for the format declaration of the EXECUTE routine.
### 4.9.3 Linking Considerations

The MAS consists of PASCAL procedures that have been compiled and inserted into an OLB/library. They may be incorporated into an application program by the appropriate data definition statement and linker control statement as follows:

```
[GMAP.V33.MASOLB]MAS3OBJ.OLB/Library
```
SECTION 5

NAME/VALUE INTERFACE

5.1 Overview

The NVI frees applications programmers from concern for the physical location of attributes for entities in the working form of the MAS. Applications programmers need only the attribute name and data type from the physical schema definition to obtain the attribute value. The benefits of the NVI include the ability to alter the physical data structure without impacting program source code, the removal of the need to program and maintain attribute data structures and access algorithms by the applications programmers, and the concentration of efficiency concerns at the system level.

The following capabilities are provided to achieve the above benefits for various commonly used high-order application languages, application environments, and host processors.

DIRECT QUERY/STORE SUBPROGRAMS, to be called by applications programs that use an attribute value for a specified entity (including an attribute for a constituent entity);

PROCEDURAL QUERY SUBPROGRAMS, to be called by applications programs that require a list of entities that have a specified attribute value (including an attribute for a constituent entity);

The Direct Query/Store and Procedural Query subprograms require translation of an attribute name into a location within the ADB, according to the physical schema definition for a particular entity KIND. A run-time subschema entry is created for those entity KINDs that are present in the working form when they are referred to by a call to the NVI. The run-time subschema defines the mapping of an attribute name to the physical location of the attribute value.

5.2 Direct Query/Store

5.2.1 Function

The Direct Query function of the NVI obtains an attribute value from the MAS working form (including an attribute of a constituent entity) for a specified entity key and attribute name. The Direct Store function replaces an attribute value in the MAS working form (including an attribute of a constituent entity) for a specified entity key and attribute name. Binding to the schema is performed at run time.

5-1
The current implementation provides for attribute data types of integer, real, string, logical, enumeration, pointer, and array.

The definition of the attribute is obtained from the schema at run time. The current implementation uses either the GMAP Data Dictionary or the Run-Time Subschema binary files produced by the GMAP Schema Manager software as the source of schema information. The scope of the NVI is limited to the attributes of entities that are defined in the copy of the schema definition used by the application for processing. It is assumed that the entity instance corresponds to the schema definition for the entity. The default GMAP Data Dictionary includes 221 GMAP entities. It may be supplemented by using the GMAP Schema Manager software to add entity definitions.

The NVI functions are designed for use by programs of any of the commonly used high-order languages (for example, the environment used for testing the NVI subprograms is a mixture of FORTRAN and PASCAL). One of the implications of this approach is that the application program that calls a NVI function must provide a data area for the attribute value that is compatible with the schema definition of the attribute (for example, the size of the data area when dealing with string data types).

The attribute name used in a call to a NVI function is an array of characters. The name is terminated by a null (a byte containing a hexadecimal "00"). When the name refers to an attribute in a constituent entity, it consists of a segment containing the name of the attribute specifying the constituency (i.e., pointer type), followed by a segment containing the name of the attribute in the constituent. The name segments are separated by a period. There may be multiple segments for specifying compound constituency. Only the last segment is terminated by a null. Trailing blanks within a segment may be omitted.

When referring to an element within an array, or to a subarray, the name of the attribute specifying the array is followed by open and close parentheses. The subscript value itself is placed in the numeric array associated with the name of the attribute. When the array is multidimensional, commas are inserted inside the parentheses. For an n-dimension array, there will be n-1 commas to specify a single element. The commas indicate the additional subscript values in the numeric array required to identify the element or the subarray. When referring to the entire array, there are no parentheses or commas.
5.2.2 Direct Query Format

NVDQAN (Entity_Key, Name_String, Subscript_Values, Attribute_Value, Return_Code)

where the data type for:

Entity_Key is an ENTKEY (input).
Name_String is a T_ATTRIBUTE_NAME; the number of characters used depends on the attribute name (input).
Subscript_Values is a T_DIMEN_VALUE; the number of entries used depends on the number of dimensions specified (input).
Attribute_Value is a T_ATTRIBUTE_VALUE (output).
Return_Code is an EXT_RET_CODE (output).

The possible return code values are:

0 = Success.
1 = Failure: the entity KIND is not defined in the run-time subschema.
2 = Failure: the attribute name is not defined for the entity in the run-time subschema.
3 = Failure: the entity key is nil (the KIND cannot be determined).
-1 = Warning: an invalid entry in the GMAP Data Dictionary was detected during the translation of an attribute; the warning message written to ddname = OUTPUT describes the error in detail.

An attribute value is obtained for the return codes for success (0) and warning (-1); no attribute value is obtained for the return codes of failure (1, 2, or 3).
5.2.3 Direct Store Format

NVDSAV (Entity_Key, Name_String, Subscript_Values, Attribute_Value, Return_Code)

where the data type for:

Entity_Key is an ENTKEY (input).
Name_String is a T_ATTRIBUTE_NAME; the number of characters used depends on the attribute name (input).
Subscript_Values is a T_DIMEN_VALUE; the number of entries used depends on the number of dimensions specified (input).
Attribute_Value is a T_ATTRIBUTE_VALUE (input).
Return_Code is an EXT_RETCODE (output).

The possible return code values are:

0 = Success.
1 = Failure: the entity KIND is not defined in the run-time subschema.
2 = Failure: the attribute name is not defined for the entity in the run-time subschema.
3 = Failure: the entity key is nil (the KIND cannot be determined).
-1 = Warning: an invalid entry in the GMAP Data Dictionary was detected during the translation of an attribute; the warning message written to ddbname = OUTPUT describes the error in detail.

An attribute value is replaced in the return codes for success (0) and warning (-1); the attribute value is not replaced for the return codes for failure (1, 2, or 3).

5.3 Procedural Query

5.3.1 Function

The Procedural Query function of the NVI evaluates an input application list of entities and creates an output application list of entities. The output is determined by a selection criterion based on an attribute of the
entities in the MAS working form (including an attribute of a constituent entity). Entities are selected based on the specified relation to the specified value for the specified attribute name. Binding to the schema is performed at run time.

The current implementation will evaluate attribute data types of integer, real, string, logical, enumeration, pointer, and array.

The definition of the attribute is obtained from the schema at run time. The current implementation uses a file in the format of the GMAP Data Dictionary or the Run-Time Subschema binary files produced by the GMAP Schema Manager software as the source of schema information. The scope of the NVI is limited to the attributes of entities that are defined in the copy of the schema definitions used by the application for processing. It is assumed that the entity instance corresponds to the schema definition for the entity. The default GMAP Data Dictionary includes 221 GMAP entities. It may be supplemented by using the GMAP Schema Manager software to add entity definitions.

The NVI functions are designed for use by programs of any of the commonly used high order languages (for example, the environment used for testing the NVI subprograms is a mixture of FORTRAN and PASCAL). One of the implications of this approach is that the application program that calls a NVI function must specify an attribute value that is compatible with the schema definition of the attribute (for example, the size of the data area when dealing with string data types).

The attribute name used in a call to a NVI function is an array of characters. The name is terminated by a null (a byte containing a hexadecimal "00"). When the name refers to an attribute in a constituent entity, it consists of a segment containing the name of the attribute specifying the constituency (i.e., pointer type), followed by a segment containing the name of the attribute in the constituent. The name segments are separated by a period. There may be multiple segments for specifying compound constituency. Only the last segment is terminated by a null. Trailing blanks within a segment may be omitted.

When referring to an element within an array, or to a subarray, the name of the attribute specifying the array is followed by open and close parentheses. The subscript value itself is placed in the numeric array associated with the name of the attribute. When the array is multidimensional, commas are inserted inside the parentheses. For an n-dimensional array there will be n-1 commas to specify a single element. The commas indicate the additional subscript values in the numeric array required to identify the element or the subarray. When referring to the entire array, there are no parentheses or commas.
5.3.2 Format

NVPQAV (Candidate_List, Name_String, Attribute_Value, Subscript_Values, Comparison_Operator, Selected_List, Return_Code)

where the data type for:

Candidate_List is a LISTKEY (input).
Name_String is a T_ATTRIBUTE_NAME; the number of characters used depends on the attribute name (input).
Attribute_Value is a T_ATTRIBUTE_VALUE (input).
Subscript_Values is a T_DIMEN_VALUE; the number of entries used depends on the number of dimensions specified (input).
Comparison_Operator is an INTEGER (input). The possible values are:
   1 = Attribute value equal
   2 = Attribute value less than
   3 = Attribute value greater than
   4 = Attribute value not equal
   5 = Attribute value less than or equal
   6 = Attribute value greater than or equal

Selected_List is a LISTKEY (output).
Return_Code is an EXT_RET_CODE (output).

The possible return code values are:

0 = Success.
1 = Failure: the entity KIND is not defined in the run-time subschema.
2 = Failure: the attribute name is not defined for the entity in the run-time subschema.
4 = Failure: the call to MAS routine MAEXEQ failed.

5 = Failure: the call to MAS routine MAL failed.

-1 = Warning: an invalid entry in the GMAP Data Dictionary was detected during the translation of an attribute; the warning message written to ddbname = OUTPUT describes the error in detail.

A valid list of entity instances is obtained for the return codes for success (0) and warning (-1); no valid list of entity instances is obtained for the return codes of failure (1, 2, 4, or 5).

5.4 Utilities

5.4.1 Function

The utility routines allow an application program to query the entity definitions at run-time. This is not normally necessary, but might be done during development testing, or if entity definitions are frequently changed, as a part of the one-time initialization for the application.

The first utility routine, NVGTAT, obtains the data type for a specified attribute. The attribute is identified by its name and the KIND number of the entity. If the attribute data type is INTEGER, REAL, or STRING, the size for the data type is also returned. If the attribute data type is POINTER, the constituent list position is also returned. The attribute name is an array of characters terminated by a null (a byte containing a hexadecimal "00"). Trailing blanks may be omitted.

The second utility routine, NVGTED, obtains (only from the Data Dictionary form of the entity definitions) the size of an entity's Application Data Block (ADB) and the length of its Constituent List (CL). The entity is identified by its name. The entity name is an array of characters terminated by a null (a byte containing a hexadecimal "00"). Trailing blanks may be omitted.

5.4.2 Attribute Data Type Query Format

NVGTAT (Entity_Kind, Attribute_Name, Data_Type, Size, Return_Code)

where the data type for:

Entity_Kind is an INTEGER; the KIND number of the entity containing the attribute definition to be queried (input).

Attribute_Name is a T_ATTRIBUTE_NAME; the number of characters used depends on the attribute name (input).
Data_Type is a T_DATA_TYPE; an enumerated scalar indicating the attribute data type (output).

Size is an INTEGER; the size for INTEGER, REAL, or STRING data types; the constituent list position for the POINTER data type; no meaning for ARRAY, LIST, SET, LOGICAL, or ENUMERATION data types (output).

Return_Code is an INTEGER (output).

The possible return code values are:
0 = Success
1 = Failure: the entity KIND is not defined in the run-time subschema
2 = Failure: the attribute name is not defined for the entity in the run-time subschema

The attribute Data_Type and Size are obtained for the return code of success (0); neither is obtained for the return codes of failure (1 or 2).

5.4.3 Entity Size Query Format

HVGTED (Entity_Name, Entity_Kind, ADB_Size, CL_Length, Return_Code)

where the data type for:

Entity_Name is a T_ATTRIBUTE_NAME; the number of characters used depends on the entity name (input).

Entity_Kind is an ORD_KIND (output).

ADB_Size is an INTEGER (output).

CL_Length is an INTEGER (output).

Return_Code is an EXT_RET_CODE (output).

The possible return code values are:
0 = Success
1 = Failure: the entity name is not defined in the run-time subschema

The Entity_Kind, ADB_Size, and CL_Length are obtained for the return code of success (0); none of them are obtained for the return code of failure (1).
5.5 IBM/MVS Environment

5.5.1 Compiling Considerations

The NVI may be used by an application program with the appropriate constants and types. For PASCAL programs, these are defined in the following INCLUDE files:

dsname = CAD5.GMAP.V33.NVIINC
member = APPLTYP

dsname = CAD5.GMAP.MASINC
member = APLTYP

5.5.2 Include Files

The types and constants used for the NVI which are contained in the NVI include file member APPLTYP or the MAS include file member APLTYP are summarized below:

Const

END_OF_STRING = 00'XC;
MAX_ATTRIBUTE_NAME = 1000;
MAX_DIMENSIONS = 100;
MAX_ENUMERATION = 16;
MAX_FIXED_STRING = 132;
MAX_VARIANT_VALUE = 1000;

Type

ANYKEY = INTEGER;
EXT_RET_CODE = INTEGER;
ORD_KIND = INTEGER;
TATTRIBUTE_NAME = ARRAY(.1..MAX_ATTRIBUTE_NAME.) OF CHAR;
T_DATA_TYPE = (INTEGER_DT, REAL_DT, STRING_DT, LOGICAL_DT,
ENUM_DT, PTR_DT, ARRAY_DT);
T_DIMENSION_VALUE = ARRAY(.1..MAX_DIMENSIONS.) OF INTEGER;
T_ENUMERATION = PACKED ARRAY(.1..MAX_ENUMERATION.) OF CHAR;
T_FIXED_STRING = PACKED ARRAY(.1..MAX_FIXED_STRING.) OF CHAR;
T_INTEGER_1 = PACKED -128..127;
T_INTEGER_2 = PACKED -32768..32767;
T_VARIANT_VALUE = ARRAY(.1..MAX_VARIANT_VALUE.) OF CHAR;
T_ATTRIBUTE_VALUE  = RECORD
CASE INTEGER OF
  0 : ( AS_VARIANT : T_variant_value );
  1 : ( AS_INTEGER_1 : T_Integer_1 );
  2 : ( AS_INTEGER_2 : T_Integer_2 );
  3 : ( AS_INTEGER_4 : Integer );
  4 : ( AS_REAL_4 : SHORTREAL );
  5 : ( AS_REAL_8 : REAL );
  6 : ( AS_FIXED_STRING : T_FIXED_STRING );
  7 : ( AS_LOGICAL : BOOLEAN );
  8 : ( AS_ENUMERATION : T_ENUMERATION );
  9 : ( AS_ENTKEY : ENTKEY );
END

The NVI INCLUDE file also contains the formal declarations for the interface routines. The member names in the INCLUDE file are the same as the interface routine names.

5.5.3 Linkage Considerations

The NVI consists of subprograms that have been processed by the linkage editor into a single module. The subprograms may be incorporated into an application program by the appropriate data definition statement and linkage editor control statement containing the following:

```
ddname = NVILIB
disp = SHR
dsnname = CAD2.GMAP.V33.LOAD

ddname = MASLIB
disp = SHR
dsnname = CAD2.GMAP.V33.LOAD
```

```
INCLUDED NVILIB(NVI)
INCLUDED MASLIB(MAS)
```

5.5.4 Processing Considerations

The NVI will automatically retrieve run-time subschema definitions from one of two possible sources: the GMAP Data Dictionary files, or the run-time Subschema binary files. The source to be used will be determined at the time the NVILIB was installed.
The files for the **Data Dictionary** are specified by:

```
  ddname = DDINX
  disp = SHR
  dsname = CAD5.GMAP.V33.DDINDX.DATA

  ddname = DDFILE
  disp = SHR
  dsname = CAD5.GMAP.V33.DDDEFN.DATA
```

The files for the **Run-Time Subschema** are specified by:

```
  ddname = INXFILE
  disp = SHR
  dsname = CAD5.GMAP.V33.RTSI

  ddname = DATAFILE
  disp = SHR
  dsname = CAD5.GMAP.V33.RTSD
```

If any messages occur during the translation from the GMAP Data Dictionary format and conventions to the run-time subschema format and conventions, they are written to:

```
  ddname = OUTPUT
  lrecl = 133
  recfm = A
```

The active portion of the run-time subschema is stored in the working form and, therefore, consumes memory. The amount for each entity KIND is:

48 bytes, plus
28 bytes * the number of attributes for the entity, plus
8 bytes * the number of attributes of enumeration data type, plus
16 bytes * the number of possible enumeration values, plus
8 bytes * the number of attributes of array data type, plus
8 bytes * the number of array dimensions, plus
8 bytes * the number of attributes of constituent reference data type, plus
4 bytes * the number of possible kinds for the constituent references.

The working form contains entries only for the KINDs of the entity instances that are specified in calls to the NVI.
5.6 VAX/VMS Environment

5.6.1 Compiling Considerations

The NVI may be used by an application program with the appropriate constants and types. For PASCAL programs, these are defined in the following INC files:

directory = [GMAP.V33.MASINC]
file name = APLTYP
directory = [GMAP.V33.NVIINC]
file name = APPLTYP

5.6.2 Include Files

The types and constants used for the NVI which are contained in the NVI include file APPLTYP or the MAS include file APLTYP ARE SUMMARIZED BELOW:

Const
MAX_ATTRIBUTE_NAME = 1000;
MAX_DIMENSIONS = 100;
MAX_ENUMERATION = 16;
MAX_FIXED_STRING = 132;
MAX_VARIANT_VALUE = 1000;

Type
ANYKEY = INTEGER;
EXT_RET_CODE = INTEGER;
ORD_KIND = INTEGER;
T_ATTRIBUTE_NAME = ARRAY(1..MAX_ATTRIBUTE_NAME) OF CHAR;
T_DATA_TYPE = (INTEGER_DT, REAL_DT, STRING_DT, LOGICAL_DT,
               ENUM_DT, PTR_DT, ARRAY_DT);
T_DIMENSION = ARRAY(1..MAX_DIMENSIONS) OF INTEGER;
T_ENUMERATION = Packed Array(1..MAX_ENUMERATION) OF CHAR;
T_FIXED_STRING = Packed Array(1..MAX_FIXED_STRING) OF CHAR;
T_INTEGER_1 = [BYTE] -128..127;
T_INTEGER_2 = [WORD] -32768..32767;
T_VARIANT_VALUE = ARRAY(1..MAX_VARIANT_VALUE) OF CHAR;

(*)
ENTKEY = ANYKEY;
LISTKEY = ANYKEY;

(*)
T_ATTRIBUTE_VALUE = RECORD

5-12
CASE INTEGER OF
  0 : ( AS_VARIANT : T_variant_value );
  1 : ( AS_INTEGER_1 : T_Integer_1 );
  2 : ( AS_INTEGER_2 : T_Integer_2 );
  3 : ( AS_INTEGER_4 : Integer );
  4 : ( AS_REAL_4 : REAL );
  5 : ( AS_REAL_8 : DOUBLE );
  6 : ( AS_FIXED_STRING : T_FIXED_STRING );
  7 : ( AS_LOGICAL : BOOLEAN );
  8 : ( AS_ENUMERATION : T_ENUMERATION );
  9 : ( AS_ENTKEY : ENTKEY );
END

Var

   END_OF_STRING : [UNSAFE] CHAR = 'XX/00';

The formal declarations for the interface routines are also contained in
the NVI include files named for the routines.

An application program needs to open a file for reading the GMAP Data
Dictionary as follows:

Open (File_Variable := DDFILE, History := READONLY, Access_Method := DIRECT,
   File_Name := / [GMAP.V33.DDFILS] GMAPDDD.DAT', Error := MESSAGE);

Open (File_Variable := DDINX, History := READONLY,
   File_Name := ' [GMAP.V33.DDFILS] GMAPDDI.DAT', Error := MESSAGE);

For PASCAL programs, these local variables need to be declared as follows:

VAR

   DDFILE : [Common]TEXT;
   DDINX : [Common]TEXT;

5.6.3 Linkage Considerations

The NVI consists of subprograms that have been compiled and inserted into
an OLB/Library. The subprograms may be incorporated into an application
program by the appropriate data definition statement and a linker control
statement containing the following:

[GMAP.V33.NVIOLB] NVI0BJ.OLB/Library

[GMAP.V33.MASOLB] MAS3OBj.OLB/Library
5.6.4 Processing Considerations

The NVI will automatically retrieve run-time subschema definitions from the GMAP Data Dictionary files.
**APPENDIX A**

MODEL ACCESS SOFTWARE (MAS) CALLING PARAMETER TYPE INDEX

<table>
<thead>
<tr>
<th>Routine</th>
<th>Calling Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAEBRST</td>
<td>(ext_ret_code)</td>
</tr>
<tr>
<td>MACPDIT</td>
<td>(anykey, namtyp, integer, ext_ret_code)</td>
</tr>
<tr>
<td>MAEA</td>
<td>(anykey, ext_ret_code)</td>
</tr>
<tr>
<td>MAEAI</td>
<td>(anykey, ext_ret_code)</td>
</tr>
<tr>
<td>MAEAV</td>
<td>(entkey, integer, ext_ret_code)</td>
</tr>
<tr>
<td>MAEC</td>
<td>(anykey, listkey, ext_ret_code)</td>
</tr>
<tr>
<td>MAECI</td>
<td>(anykey, listkey, ext_ret_code)</td>
</tr>
<tr>
<td>MAECIK</td>
<td>(anykey, ord_kind, listkey, ext_ret_code)</td>
</tr>
<tr>
<td>MAECMP</td>
<td>(entkey, listkey, ext_ret_code)</td>
</tr>
<tr>
<td>MAECQY</td>
<td>(entkey, entkey, integer, ext_ret_code)</td>
</tr>
<tr>
<td>MAECR</td>
<td>(entblock, anykey, entkey, ext_ret_code)</td>
</tr>
<tr>
<td>MAECRN</td>
<td>(entdata, anykey, entkey, integer, ext_ret_code)</td>
</tr>
<tr>
<td>MAECTK</td>
<td>(integer, ext_ret_code)</td>
</tr>
<tr>
<td>MAECXQ</td>
<td>(anykey, blkdata, routine, listkey, ext_ret_code, ext_ret_code)</td>
</tr>
<tr>
<td>MAED</td>
<td>(anykey, listkey, ext_ret_code)</td>
</tr>
<tr>
<td>MAEDI</td>
<td>(anykey, listkey, ext_ret_code)</td>
</tr>
<tr>
<td>MAEDT</td>
<td>(anykey, listkey, listkey, ext_ret_code)</td>
</tr>
<tr>
<td>MAEDTI</td>
<td>(anykey, listkey, listkey, ext_ret_code)</td>
</tr>
<tr>
<td>MAEGKN</td>
<td>(entkey, integer, ext_ret_code)</td>
</tr>
<tr>
<td>MAEGTK</td>
<td>(entkey, entblock, ext_ret_code)</td>
</tr>
<tr>
<td>MAEKNRD</td>
<td>(integer, ord_kind, ext_ret_code)</td>
</tr>
<tr>
<td>MAERST</td>
<td>(namtyp, ext_ret_code)</td>
</tr>
<tr>
<td>MAESCI</td>
<td>(anykey, integer ext_ret_code)</td>
</tr>
<tr>
<td>MAESVL</td>
<td>(entkey, integer, ext_ret_code)</td>
</tr>
<tr>
<td>MAESWA</td>
<td>(ext_ret_code)</td>
</tr>
<tr>
<td>MAESWT</td>
<td>(anykey, integer, ext_ret_code)</td>
</tr>
<tr>
<td>MAEU</td>
<td>(anykey, listkey, ext_ret_code)</td>
</tr>
<tr>
<td>MAEUD</td>
<td>(entkey, entblock, ext_ret_code)</td>
</tr>
<tr>
<td>MAEUI</td>
<td>(anykey, listkey, ext_ret_code)</td>
</tr>
<tr>
<td>MAEUUK</td>
<td>(anykey, ord_kind, listkey, ext_ret_code)</td>
</tr>
<tr>
<td>MAEUSR</td>
<td>(entkey, integer, ext_ret_code)</td>
</tr>
<tr>
<td>MAEXUQ</td>
<td>(anykey, blkdata, routine, listkey, ext_ret_code, ext_ret_code)</td>
</tr>
<tr>
<td>MAEXEQ</td>
<td>(anykey, blkdata, routine, ext_ret_code, ext_ret_code)</td>
</tr>
<tr>
<td>MAINIT</td>
<td>(ext_ret_code)</td>
</tr>
<tr>
<td>MAKCNT</td>
<td>(integer, integer, ext_ret_code)</td>
</tr>
<tr>
<td>MAKILL</td>
<td>(ext_ret_code)</td>
</tr>
<tr>
<td>MAKEEQ</td>
<td>(anykey, variant, entry point, integer, integer)</td>
</tr>
</tbody>
</table>
**Routine** | **Calling Sequence**
--- | ---
MAL | (listkey, ext_ret_code)
MALAND | (anykey, anykey, listkey, ext_ret_code)
MALATC | (anykey, anykey, ext_ret_code)
MALCPY | (listkey, listkey, ext_ret_code)
MALD | (listkey, ext_ret_code)
MALDA | (ext_ret_code)
MALDI | (anykey, ext_ret_code)
MALFND | (anykey, entkey, integer, integer, ext_ret_code)
MALGTK | (anykey, integer, entkey, ext_ret_code)
MALINS | (anykey, anykey, integer, ext_ret_code)
MALK | (ord_kind, listkey, ext_ret_code)
MALKC | (anykey, ord_kind, listkey, ext_ret_code)
MALKL | (anykey, ord_kind, listkey, ext_ret_code)
MALKU | (anykey, ord_kind, listkey, ext_ret_code)
MALN | (integer, listkey, ext_ret_code)
MALNO | (anykey, integer, ext_ret_code)
MALNOT | (anykey, anykey, listkey, ext_ret_code)
MALOCK | (listkey, integer, ext_ret_code)
MALOR | (anykey, anykey, listkey, ext_ret_code)
MALRD | (anykey, entkey, ext_ret_code)
MALDE | (listkey, ext_ret_code)
MALREP | (anykey, anykey, ext_ret_code)
MALRMV | (anykey, integer, ext_ret_code)
MALK (MALROR) | (anykey, ext_ret_code)
MALRPL | (anykey, entkey, integer, ext_ret_code)
MALRI | (anykey, ext_ret_code)
MALRSTI | (listkey, integer)
MALRV | (anykey, ext_ret_code)
MALSRT | (anykey, routine, ext_ret_code)
MALSTF | (anykey, ext_ret_code)
MALSTR | (anykey, ext_ret_code)
MALXEQ | (anykey, blkdata, routine, listkey, ext_ret_code, ext_ret_code)
MAQURY | (entkey, namtyp, integer, ext_ret_code)
MARDLT | (ord_kind, integer)
MARSMT | (ord_kind, T_schema_pointer, integer)
MASNSZ | (integer, integer, ext_ret_code)
MAUDP | (anykey, namtyp, integer, ext_ret_code)
MIDD | (anykey, integer)
MIDBR | (anykey, position, integer)
MRSCR | (ord_kind, integer, T_schema_pointer, integer)
# APPENDIX B

## ALPHABETICAL MODEL ACCESS SOFTWARE (MAS) ROUTINE INDEX

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MABRST</td>
<td>Reset process and application flags</td>
</tr>
<tr>
<td>MACDPT</td>
<td>Update Constituent SYSUSE flag</td>
</tr>
<tr>
<td>MAEA</td>
<td>Activate an entity or list of entities</td>
</tr>
<tr>
<td>MAEAI</td>
<td>Activate an entity or list of entities inclusively</td>
</tr>
<tr>
<td>MAEAV</td>
<td>Find value of entity activation setting</td>
</tr>
<tr>
<td>MAEC</td>
<td>Create list of constituents</td>
</tr>
<tr>
<td>MAECI</td>
<td>Create list of inclusive constituents</td>
</tr>
<tr>
<td>MAECIK</td>
<td>Create list of inclusive constituents by kind</td>
</tr>
<tr>
<td>MAECMP</td>
<td>Create a list of constituents that compress</td>
</tr>
<tr>
<td>MAECQY</td>
<td>Determine if user compresses a constituent</td>
</tr>
<tr>
<td>MAECR</td>
<td>Create an entity</td>
</tr>
<tr>
<td>MAECRN</td>
<td>Create entity with constituent list size</td>
</tr>
<tr>
<td>MAECTK</td>
<td>Get number of different kinds in working-form model</td>
</tr>
<tr>
<td>MAEXQ</td>
<td>Process constituents via an application defined procedure</td>
</tr>
<tr>
<td>MAED</td>
<td>Delete an entity or list of entities</td>
</tr>
<tr>
<td>MAEDI</td>
<td>Delete an entity or list of entities inclusively</td>
</tr>
<tr>
<td>MAEDT</td>
<td>Delete test an entity or list of entities</td>
</tr>
<tr>
<td>MAEDTI</td>
<td>Delete test an entity or list of entities inclusively</td>
</tr>
<tr>
<td>MAEGKN</td>
<td>Get kind value of an entity</td>
</tr>
<tr>
<td>MAEGTK</td>
<td>Get entity ADB</td>
</tr>
<tr>
<td>MAEKND</td>
<td>Get kind value at specified position in kind list</td>
</tr>
<tr>
<td>MAERST</td>
<td>Set application flag in all entities in model to &quot;off&quot;</td>
</tr>
<tr>
<td>MAESCI</td>
<td>Set or reset process flag for inclusive constituents</td>
</tr>
<tr>
<td>MAESVL</td>
<td>Find binary switch setting of an entity</td>
</tr>
<tr>
<td>MAESWA</td>
<td>Set all entities binary switch setting to &quot;off&quot;</td>
</tr>
<tr>
<td>MAESWT</td>
<td>Set binary switch in an entity or list of entities</td>
</tr>
<tr>
<td>MAEU</td>
<td>Create list of users</td>
</tr>
<tr>
<td>MAEUD</td>
<td>Update entity ADB</td>
</tr>
<tr>
<td>MAEUI</td>
<td>Create list of users inclusively</td>
</tr>
<tr>
<td>MAEUIK</td>
<td>Create list of users inclusively by kind</td>
</tr>
<tr>
<td>MAEUR</td>
<td>Determine if an entity has any users</td>
</tr>
<tr>
<td>MAEXUQ</td>
<td>Process users via an application defined procedure</td>
</tr>
<tr>
<td>MAEXEQ</td>
<td>Execute procedure on an entity or list of entities</td>
</tr>
<tr>
<td>MAINIT</td>
<td>Initialize the working-form model</td>
</tr>
<tr>
<td>MAKCNT</td>
<td>Determine number of entities in model with specified kind</td>
</tr>
<tr>
<td>MAKILL</td>
<td>Delete the current working-form model</td>
</tr>
<tr>
<td>MAKXEQ</td>
<td>Execute procedure on all entities of specified kind</td>
</tr>
<tr>
<td>MAL</td>
<td>Create an empty list</td>
</tr>
<tr>
<td>MALAND</td>
<td>&quot;And&quot; of two list</td>
</tr>
</tbody>
</table>

B-1
### Routine Description

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MALATC</td>
<td>Attach entity or list of entities to entity or list</td>
</tr>
<tr>
<td>MALCPY</td>
<td>Make a copy of a list</td>
</tr>
<tr>
<td>MALD</td>
<td>Delete a list</td>
</tr>
<tr>
<td>MALDA</td>
<td>Delete all lists in the working-form model</td>
</tr>
<tr>
<td>MALDI</td>
<td>Delete a list and all lists after it</td>
</tr>
<tr>
<td>MALFND</td>
<td>Find position of an entity in a list</td>
</tr>
<tr>
<td>MALGETK</td>
<td>Get the Nth entity from a list</td>
</tr>
<tr>
<td>MALINS</td>
<td>Insert entity or list of entities into a list</td>
</tr>
<tr>
<td>MALK</td>
<td>Create list of an entities of specified kind</td>
</tr>
<tr>
<td>MALKCI</td>
<td>Create list of entities of a kind from constituents of another list</td>
</tr>
<tr>
<td>MALKLI</td>
<td>Create list of entities of specified kind which are found within another list</td>
</tr>
<tr>
<td>MALKUI</td>
<td>Create list of entities of a kind from users of another list</td>
</tr>
<tr>
<td>MALN</td>
<td>Create an empty list of specified size</td>
</tr>
<tr>
<td>MALNO</td>
<td>Count entities in a list</td>
</tr>
<tr>
<td>MALNOT</td>
<td>&quot;Not&quot; of two lists</td>
</tr>
<tr>
<td>MALOCK</td>
<td>Set the list lock flag</td>
</tr>
<tr>
<td>MALOR</td>
<td>&quot;Or&quot; of two lists</td>
</tr>
<tr>
<td>MALRD</td>
<td>Read next entry in list</td>
</tr>
<tr>
<td>MALRDE</td>
<td>Remove duplicate entities from list</td>
</tr>
<tr>
<td>MALREP</td>
<td>Replace list of entities</td>
</tr>
<tr>
<td>MALRMV</td>
<td>Remove entity or list of entities</td>
</tr>
<tr>
<td>MALROR</td>
<td>Sort entities in direct user to constituent order</td>
</tr>
<tr>
<td>MALRPL</td>
<td>Replace entity or list of entities</td>
</tr>
<tr>
<td>MALRRI</td>
<td>Sort entities in inclusive user to constituent order</td>
</tr>
<tr>
<td>MALRST</td>
<td>Reset an application list</td>
</tr>
<tr>
<td>MALRVS</td>
<td>Reverse the order of a list</td>
</tr>
<tr>
<td>MALRST</td>
<td>Sort entities via an application defined procedure</td>
</tr>
<tr>
<td>MALSTF</td>
<td>Set flag to read in forward direction</td>
</tr>
<tr>
<td>MALSTR</td>
<td>Set flag to read in reverse direction</td>
</tr>
<tr>
<td>MALXEQ</td>
<td>Execute procedure on entity or list of entities</td>
</tr>
<tr>
<td>MAQURY</td>
<td>Determine value of application flag for given entity</td>
</tr>
<tr>
<td>MARDLT</td>
<td>Delete the run-time schema for a given entity kind</td>
</tr>
<tr>
<td>MARSST</td>
<td>Retrieve the run-time subschema for a given entity kind</td>
</tr>
<tr>
<td>MASMSSZ</td>
<td>Find actual model used space and model free space</td>
</tr>
<tr>
<td>MAUPDT</td>
<td>Update value of application flag of entity or list of entities</td>
</tr>
<tr>
<td>MIDBD</td>
<td>Delete an entity or list of entities, but do not consider the delete rules</td>
</tr>
</tbody>
</table>
**APPENDIX B (contd.)**

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDBRV</td>
<td>Remove an entity from the constituent list or remove an entity from a list of entities. Delete if marked for delete.</td>
</tr>
<tr>
<td>MRSCR</td>
<td>Create a run-time subschema for a given entity kind</td>
</tr>
</tbody>
</table>

B-3
## APPENDIX C

MODEL ACCESS SOFTWARE (MAS) RETURN CODE INDEX

<table>
<thead>
<tr>
<th>Error Type</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO_ERRORS_DETECTED</td>
<td>0</td>
</tr>
<tr>
<td>BAD_ENT_KIND</td>
<td>1</td>
</tr>
<tr>
<td>INVALID_CREATE</td>
<td>2</td>
</tr>
<tr>
<td>CANT_CREATE_LIST</td>
<td>3</td>
</tr>
<tr>
<td>MAS_INIT_FAILED</td>
<td>4</td>
</tr>
<tr>
<td>INVALID_UPDATE</td>
<td>5</td>
</tr>
<tr>
<td>CANT_UPDATE_ENT</td>
<td>6</td>
</tr>
<tr>
<td>CANT_CREATE_ENT</td>
<td>7</td>
</tr>
<tr>
<td>CANT_VERIFY_CONNECT</td>
<td>8</td>
</tr>
<tr>
<td>INVALID_CONNECTION</td>
<td>9</td>
</tr>
<tr>
<td>CANT_CONNECT</td>
<td>10</td>
</tr>
<tr>
<td>ABSENT_INPUT</td>
<td>11</td>
</tr>
<tr>
<td>INVALID_GET</td>
<td>12</td>
</tr>
<tr>
<td>NDS_OP_COMPLETE</td>
<td>13</td>
</tr>
<tr>
<td>BAD_LIST_POSITION</td>
<td>14</td>
</tr>
<tr>
<td>MAXIMUM_LIST_SIZE</td>
<td>15</td>
</tr>
<tr>
<td>BAD_LIST_MOVE_COUNT</td>
<td>16</td>
</tr>
<tr>
<td>BAD_LIST_REFERENCE</td>
<td>17</td>
</tr>
<tr>
<td>BAD_ENT_KEY</td>
<td>18</td>
</tr>
<tr>
<td>DUPLICATE_SCH</td>
<td>19</td>
</tr>
<tr>
<td>DUMP_ERROR</td>
<td>20</td>
</tr>
<tr>
<td>BAD_ENT_SIZE</td>
<td>21</td>
</tr>
<tr>
<td>BAD_SCH_KIND</td>
<td>22</td>
</tr>
<tr>
<td>PROC_CODE_ERROR</td>
<td>23</td>
</tr>
<tr>
<td>PROC_OUT_OF_RANGE</td>
<td>24</td>
</tr>
<tr>
<td>NO_MATCH_FOUND</td>
<td>25</td>
</tr>
<tr>
<td>DUPS_NOT_REMOVED</td>
<td>26</td>
</tr>
<tr>
<td>INVALID_DELETE</td>
<td>27</td>
</tr>
<tr>
<td>BAD_ENTITY_ON_USER_LIST</td>
<td>28</td>
</tr>
<tr>
<td>BAD_DELETE_KEY</td>
<td>29</td>
</tr>
<tr>
<td>EMPTY_MODEL</td>
<td>30</td>
</tr>
<tr>
<td>ARG_OUT_OF_RANGE</td>
<td>31</td>
</tr>
<tr>
<td>INVALID_CRB_POSITION</td>
<td>32</td>
</tr>
<tr>
<td>CRB_ENTRY_NOT_FOUND</td>
<td>33</td>
</tr>
<tr>
<td>INVALID_FLAG_NAME</td>
<td>34</td>
</tr>
<tr>
<td>CANT_MARK_ENTITY_DELETE</td>
<td>35</td>
</tr>
<tr>
<td>SIZE_NOT_CARE_ENOUGH</td>
<td>36</td>
</tr>
<tr>
<td>RTS_NOT_IN_WORKING_FORM</td>
<td>37</td>
</tr>
<tr>
<td>CORE_NOT_AVAILABLE</td>
<td>38</td>
</tr>
<tr>
<td>NOT_ENOUGH_CORE_FOR_INIT</td>
<td>39</td>
</tr>
<tr>
<td>ABSOLUTELY_NO_MORE_CORE</td>
<td>40</td>
</tr>
</tbody>
</table>

C-1
APPENDIX C (contd.)

<table>
<thead>
<tr>
<th>Error type</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAINIT_ALREADY_DONE</td>
<td>41</td>
</tr>
<tr>
<td>ROLE_DOES_NOT_MATCH</td>
<td>42</td>
</tr>
<tr>
<td>ENTITY_NOT_FOUND_LIST</td>
<td>43</td>
</tr>
</tbody>
</table>
## APPENDIX C (contd.)

### MODEL ACCESS SOFTWARE RETURN CODE INDEX

<table>
<thead>
<tr>
<th>Warning type</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>OKW</td>
<td>0</td>
</tr>
<tr>
<td>NO_SUCH_SCH</td>
<td>-1</td>
</tr>
<tr>
<td>PROC_WARNING_CODE</td>
<td>-2</td>
</tr>
<tr>
<td>EMPTY_DELETE_LIST</td>
<td>-3</td>
</tr>
<tr>
<td>EMPTY_EXCEPTION_LIST</td>
<td>-4</td>
</tr>
<tr>
<td>END_OF_LIST</td>
<td>-5</td>
</tr>
<tr>
<td>NO_LIST CREATED</td>
<td>-6</td>
</tr>
<tr>
<td>EMPTY_MARK_LIST</td>
<td>-7</td>
</tr>
<tr>
<td>NO_LIST GIVEN</td>
<td>-11</td>
</tr>
</tbody>
</table>

C-3
APPENDIX D
GENERAL TECHNIQUES/GUIDELINES

- Avoid creating long lists of entities:
  - Lists are processed sequentially
  - Lists use model space

- Do not use ENTKEY as a memory address:
  - ENTKEY does not address the attribute data block of the entity

- Avoid "nil" keys:
  - Abend or nil pointer checking errors may be caused

- Delete application lists when no longer needed:
  - Application lists use memory
  - Application lists slow deletion of entities

- Always test the Model Access Software (MAS) interface return code:
  - RC = 0  normal return
  - RC < 0  warning message
  - RC > 0  critical error

- Reset the process bit to "off" when it is no longer needed.

- Define the KIND and LENGTH fields in the ADB.

- When MALRD is used in conjunction with one of the following interface routines:
  
  MAED    MALINS
  MAEDI   MALRMV
  MAL

  the position of sequential reading is incremented/decremented if an
  interface function modifies the list.

  Do not use MALGTK and one of the above routines because the local
  variable position cannot be adjusted by the MAS package.
APPENDIX D (contd.)

For example:

VAR NUM_IN_LIST: = INTEGER

BEGIN

FOR I = 1 TO NUM_IN_LIST DO

MALGTK (LISTKEY, NUM_IN_LIST, ENTKEY1): 

MAED (ENTKEY1, LISTX):

END:

As each entity is deleted, it is removed from the LISTKEY list, but I is not adjusted.

o With the exception of MAL and MALK, empty lists will not be created. If an interface function has an output LISTKEY and the list is empty, the list will not be created and the LISTKEY will be NIL. A warning return code will indicate this situation.
INTRODUCTION

The Model Access Software (MAS) consists of a set of PASCAL procedures that provides an interface to the working form model for application programs. When the application programs are written in a language other than PASCAL, the run-time environment must satisfy the interlanguage communication requirements of all the languages involved. This appendix discusses the MAS interlanguage environment conventions and the composition of the PASCAL dynamic storage areas. Examples are given for a FORTRAN program that uses MAS routines.

INTERLANGUAGE CONVENTIONS

When the MAS subprograms were compiled, they were defined as PROCEDUREs using SUBPROGRAM declarations. The subprogram declaration is an extension to IBM Pascal that allows a PASCAL procedure to be called from any language. The subprogram declaration supplies special code at compile time. At run-time, this code determines the nature of the calling program. For non-PASCAL calls, two macros are invoked: Prolog and Epilog. Before the procedure executes, Prolog locates the PASCAL Communication Work Area (PCWA) as well as the main and local Dynamic Storage Areas (DSA) and establishes the PASCAL register conventions. On exit, the Epilog macro restores the register conventions of the calling program.

The effect of this method is that no special action is required by the calling program, regardless of its language.

The SUBPROGRAM declaration may also be applied to application procedures, which may then be called from, and make calls to, routines of any language. This method is limited to PASCAL PROCEDURES and does not apply to PASCAL FUNCTIONS.

ESTABLISHING INTERLANGUAGE ENVIRONMENT

The preferred (and easiest) approach is to insert the entire application into a PASCAL program. This method, shown in Figure E-1, assures correct error handling.
Figure E-1. PASCAL Environment

An alternate approach, illustrated in Figure E-2, is to insert the portion of the application that makes the MAS calls into a PASCAL procedure that is declared MAIN. The error handling capability, however, may be limited in this method. Note that the model created within the scope of the MAIN PASCAL procedure is active only during the execution of the MAIN procedure; new models may be created in subsequent calls to a similarly declared MAIN procedure. Upon termination of the last call to a PASCAL MAIN, the procedure PSCLHX should be called to terminate the PASCAL run-time environment.

Examples of the PASCAL source and link-edit instructions are included at the end of this appendix. Figure E-3 illustrates the PASCAL dynamic storage area stack.
Figure E-2. MAIN Procedure
REGISTER CONVENTIONS

The interlanguage environment establishes the correct register conventions automatically. The following information is included for use from the IBM TEST mode.

<table>
<thead>
<tr>
<th>Register</th>
<th>PASCAL</th>
<th>Non-PASCAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>Branch address</td>
<td>Branch address</td>
</tr>
<tr>
<td>14</td>
<td>Return address</td>
<td>Return address</td>
</tr>
<tr>
<td>13</td>
<td>Local DSA address (1)</td>
<td>Save area address</td>
</tr>
<tr>
<td>12</td>
<td>PCWA address</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Main DSA address</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Address of parameter</td>
<td>Address of parameter</td>
</tr>
<tr>
<td></td>
<td>list (2) (3)</td>
<td>list</td>
</tr>
<tr>
<td>0</td>
<td>(2)</td>
<td>Function value</td>
</tr>
</tbody>
</table>

NOTES:  
(1) The save area is the first entry in the local DSA, which is established by a PASCAL caller.

(2) The function value for PASCAL is referred to by the first entry in the parameter list. PASCAL input parameters for a function are referred to as starting with the second entry in the parameter list.

(3) The parameter list contains addresses of parameters except for pass-by-value of scalars, pointers, or sets, in which case the parameter list contains the actual value.

PASCAL DYNAMIC STORAGE AREA

The dynamic storage area of the PASCAL main program contains global variables (including any commons). Each PASCAL procedure invoked has a local dynamic storage area containing local variables. The dynamic storage areas are contained in a LIFO stack.

In general, the DSA of a routine consists of five sections:

(1) The local save area (144).

(2) Parameters passed in by the caller.

(3) Local variables required by the routine.

(4) A save area required by any routine that will be called.

(5) Storage for the largest parameter list to be built for a call.
Sections 1 and 2 are allocated by the calling routine; Sections 3, 4, and 5 are allocated by the Prolog of the called routine.

Every DSA is at least 144 bytes long. This is the storage required by PASCAL/VS for a save area. The local variables and parameters of the routine are mapped within the DSA starting at offset 144.

Upon entering a routine, Register 1 points 144 bytes into the DSA of the routine, which is where the parameters passed in by the caller reside.

Upon invocation, Register 13 points to the base of the DSA of the caller, which is where the save area of the caller is located. Figure E-3 illustrates the condition of the stack and relevant registers immediately upon the start of the routine.

![Figure E-3. PASCAL Dynamic Storage Area Stack](image-url)
EXAMPLES

EXAMPLE 1: PASCAL PROGRAM (PASMAIN) THAT INVOKES FORTRAN MAIN

PASCAL PROGRAM
PROGRAM PASMAIN;
PROCEDURE MAIN; FORTRAN;
BEGIN
MAIN;
END.

LINKEDIT INSTRUCTIONS
INCLUDE APLLIB(PASMAIN)
INCLUDE APLLIB(APL )
INCLUDE MASLIB(MAS )
ENTRY PASMAIN

where SYSLIB allocation includes SYS1.PASCLIB.

EXAMPLE 2: PASCAL PROCEDURE (PASSUB) INVOKED BY FORTRAN MAIN THAT INVOKES FORTRAN SUBROUTINE (FORSUB)

PASCAL PROCEDURE
SEGMENT PASSUB;
PROCEDURE PASSUB (....);MAIN; FORTRAN MAIN may pass parameters to the PASCAL subroutine.
PROCEDURE PASSUB;
PROCEDURE FORSUB(....);FORTRAN; PASCAL MAIN may pass parameters to the FORTRAN MAIN.
BEGIN
FORSUB(....);
Invokes FORTRAN subroutine that calls MAS.
end;

FORTRAN MAIN PROGRAM
:
.
CALL PASSUB(....)
CALL PSCLHX
:
.

E-6
LINKEDIT INSTRUCTIONS

INCLUDE APLLIB(APL )
List of objects including FORTRAN MAIN
or LOAD module including FORTRAN MAIN.

INCLUDE APLLIB(PASSUB)
INCLUDE MASLIB(MAS )
ENTRY APL
NAME APL

where SYSLIB allocation includes SYS1.PASCLIB.
APPENDIX F
SAMPLE PROGRAMS

INTRODUCTION

A series of sample programs, presented in Table F-1, were written by an MDC development programmer to emulate the MDC CAD system. They are reprinted here in hopes that other programmers can gain insight into the easiest and most efficient way to use each of the Model Access Software routines.

The routine descriptions in the earlier parts of this manual are cross-referenced to the sample program in which the Model Access Software routine is used (routines are all written in PASCAL):

TABLE F-1
SAMPLE PROGRAMS

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>MAS ROUTINES USED</th>
<th>APPLICATION ROUTINE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defines System Type and Constant Declarations.</td>
<td>ENTTYP</td>
<td></td>
</tr>
<tr>
<td>Entity Type Definitions.</td>
<td>ENTDEF</td>
<td></td>
</tr>
<tr>
<td>Model a Line for Display.</td>
<td>MAL, MALATC, MAEGR</td>
<td>MODLN</td>
</tr>
<tr>
<td>Retrieve Entity's Type, Special Code, and Label.</td>
<td>MAEGTK</td>
<td>CADENT</td>
</tr>
<tr>
<td>Retrieve Entity Key and Data from Pick List.</td>
<td>MALNO, MALGTK</td>
<td>PLQRY</td>
</tr>
</tbody>
</table>

F:1
TABLE F-1 (contd.)

<table>
<thead>
<tr>
<th>Implementation of Delayed &quot;Delete&quot; Operation.</th>
<th>MAEGTK, MAECI, MALNO, MAEU, MALGTK</th>
<th>RMBLK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reject Changes to Attributes.</td>
<td>MAECR, MAL, MALRTC</td>
<td>VCASAV</td>
</tr>
<tr>
<td>Checks if the KIND is between a high and a low boundary.</td>
<td>MALXEQ</td>
<td>KNDRNG</td>
</tr>
<tr>
<td>Retrieves the radius of a PRS.</td>
<td>MAEXEQ</td>
<td>R$RCRD</td>
</tr>
</tbody>
</table>

F-2
ENTTYP SAMPLE PROGRAM

ENTTYPE - Sample program defines system type and constant declarations.

TYPE
UNSIGNED_INT1 = PACKED 0..255;
SIGNED_INT1 = PACKED -128..127;
UNSIGNED_INT2 = PACKED 0..65535;
SIGNED_INT2 = PACKED -32768..32767;
COORD = (X,Y,Z);
VECTOR = ARRAY(.COORD.) OF REAL;
SHVTVECTOR = ARRAY(.COORD.) OF SHORTREAL;
TWO_PNTS = ARRAY(.1..2.) OF VECTOR;
INT2 = ARRAY(.1..2.) OF INTEGER;
INT3 = ARRAY(.1..3.) OF INTEGER;
INT4 = ARRAY(.1..4.) OF INTEGER;
SHTRREAL2 = ARRAY(.1..2.) OF SHORTREAL;
SHTRREAL3 = ARRAY(.1..3.) OF SHORTREAL;
REAL2 = ARRAY(.1..2.) OF REAL;
REAL3 = ARRAY(.1..3.) OF REAL;
CHAR2 = PACKED ARRAY(.1..2.) OF CHAR;
CHAR4 = PACKED ARRAY(.1..4.) OF CHAR;
CHAR6 = PACKED ARRAY(.1..6.) OF CHAR;
CHAR8 = PACKED ARRAY(.1..8.) OF CHAR;
PDDI Access Software_ENTITY_TYPE = INTEGER;
CADD_ENTITY_TYPE = INTEGER;
DATA_TYPE = INTEGER;
ENTDEF SAMPLE PROGRAM

ENTDEF - Sample program shows entity type definitions.

TYPE
  ANYKEY = INTEGER;
  LISTKEY = ANYKEY;
  ENTKEY = ANYKEY;
  EXT_RET_CODE = INTEGER;
  ENT_KIND = INTEGER;
  ORD_KIND = INTEGER;

T_SYS = (CADD, IDBPDDI Access Software);

T_HEADER = RECORD
  KIND : ENT_KIND;
  SIZE : 0..4194303;
  LABEL : CHAR8; --> OVER
  DSP_TYPE : INTEGER;
  SUBTYPE : INTEGER;
  VERSION : INTEGER;
END;

(*
ENTITY DATA RECORDS
*)

PNTDATA = RECORD
  UDB : T_HEADER;
  PT : VECTOR;
END;

LINDATA = RECORD
  UDB : T_HEADER;
END;

ARCDATA = RECORD
  UDB : T_HEADER;
  MIDPNT : VECTOR;
END;

CRLDATA = RECORD
  UDB : T_HEADER;
  PT1 : VECTOR;
  PT2 : VECTOR;
  PT3 : VECTOR
END;

F.4
ENTDEF SAMPLE PROGRAM (CONTINUED)

PLNDATA = RECORD
  UDB : T_HEADER;
  SYMBOL : VECTOR;
  NORMAL : VECTOR;
END;

PICDATA = RECORD
  UDB : T_HEADER;
  PICKPNT : SHIVECTOR;
END;

ENTBLOCK = RECORD
  CASE ENTITY_TYPE OF
    POINT : (PNT : PNTDATA);
    LINE : (LIN : LINDATA);
    PLANE : (PLN : PLNDATA);
    ARC : (ARC : ARCDATA);
    CIRCLE : (CRL : CRLDATA);
    PICK_ENTITY : (PIC : PICDATA);
  END;

MODLN SAMPLE PROGRAM

MODLN - Sample program to model a line for display.

CA DESCRIPTION OF ARGUMENTS
C INPUT
C PNTRS - AN ARRAY CONTAINING THE CORRELATION OF
THE START AND END POINTS OF THE LINE
C DSPTYPE - CADD DISPLAY TYPE
C = 1, SOLID LINE
C = 2, DASHED LINE
C OUTPUT
C IRC - RETURN CODE
C = 0, NORMAL RETURN
C = 18, INVALID INPUT DATA
C = 34, ERROR IN CREATION
C COMMONS
C DGRPS2 - PROVIDE SYSTEM WORK AREAS
*)
REF
DGRPS2 : T_DGRPS2;
CONST
TICKMARK = 99;
TYPE
LIN_SPCODE = (NORMAL_LINE, TICK_MARK);
LIN_DSPTYPE = (DUMMY, SOLID, DASH, CENTER, PHANTOM);
VAR
SPECIAL_CODE : INTEGER;
DISPLAY_TYPE : INTEGER;
I : INTEGER;
NEW_LINE : ENTBLOCK;
NEW_LINE_LABEL : CHAR8;
KEYLC : LISTKEY;
KEYE_LN : LISTKEY;
MODLN SAMPLE PROGRAM (CONTINUED)

BEGIN (* MODLN STARTS HERE *)

(*) CHECK IF DUPLICATE POINTS (*)

IF (PNTRS(.1.) = PNTRS(.2.))
THEN
IRC := 18
ELSE
BEGIN
(* CONVERT CADD TYPE DESIGNATION TO PDDI Access Software EQUIV *)
CVTPSC(CADD_LINE, SPECIAL_CODE, NEW_LINE.LIN.UDB.KIND,
       NEW_LINE.LIN.UDB.SUBTYPE, IDBPDDI Access Software, IRC);

(* GET A LABEL FOR THE NEW LINE *)
LDLABL(CADD_LINE, SPECIAL_CODE, NEW_LINE_LABEL, IRC);

IF IRC = 0
THEN
BEGIN
(* LOAD THE LINE BLOCK *)
WITH NEW_LINE.LIN.UDB DO
BEGIN
SIZE := UDDSIZ(PDDI Access Software LINE); (* SIZE OF ENTITY BLOCK *)
LABEL := NEW_LINE_LABEL;
DSP_TYPE := DISPLAY_TYPE;
END;

(* CREATE LINE WITH CONSL *)
MAL(KEYLC, IRC); (*CREATE EMPTY LIST*)
FOR I := 1 TO 2 DO
MALATC(KEYLC, PNTRS(.I.), IRC); (* ADD EACH END POINT TO LIST *)
MAECR(NEW_LINE, KEYLC, KEYE_LN, IRC); (* MODEL THE ENTITY *)
IF IRC = 0
THEN
BEGIN
(* RECORD THE CREATE FOR VERSION CONTROL *)
VCCREA(KEYE_LN, IRC);
END;(* END OF CHECKING IRC FROM "LDLABL *)
END;(* END OF CHECKING DUPLICATE POINTERS *)
END;(* END OF MODLN*)
CADENT SAMPLE PROGRAM

CADENT – Sample program to retrieve the type, special code, and label of an entity.

CA DESCRIPTION OF ARGUMENTS
C INPUT
C
C KEYE – KEY OF THE ENTITY
C
C OUTPUT
C ENT_TYPE – ENTITY TYPE
C SP_CODE – SPECIAL CODE
C ENTLABEL – ENTITY LABEL
C *

VAR
I, IRC : INTEGER;
ENTITY : ENTBLOCK;

BEGIN (* CADENT STARTS HERE *)
(* RETRIEVE THE ENTITY ATTRIBUTE BLOCK FROM PDDI Software *)
MAEGTK(KEYE,ENTITY,IRC);
(* TRANSFORM PDDI Access Software KIND TO TYPE AND SPECIAL CODE *)
CVTPSC(ENTITY.PNT.UDB.KIND, ENTITY.PNT.UDB.SUBTYPE,
ENT_TYPE, SP_CODE, CADD, IRC);
(* COPY LABEL OUT OF ENTITY BLOCK *)
LABEL(.I.) := ENTITY.PNT.UDB.LABEL;
END;(*END OF CADENT *)
PLQRY SAMPLE PROGRAM

PLQRY - Sample program retrieves an entity key and data from the pick list.

CA  DESCRIPTION OF ARGUMENTS
C     INPUT
C     NUMPK  - NUMBER OF PICKS DESIRED
C     OUTPUT
C     PIC_ENTITY  - ATTRIBUTES OF THE PICK ENTITY
C     PICKED_ENTITY - ATTRIBUTES OF THE PICKED ENTITY
C     IRC  - RETURN CODE
C     = 0, NORMAL RETURN
C     = 18, INVALID INPUT DATA
*)

VAR
    PICK_LIST  : LISTKEY;
    PICK_COUNT  : INTEGER;

%PAGE

BEGIN (* PLQRY STARTS HERE *)
(*
    RETRIEVE THE PICK LIST AND COUNT THE NO OF PICKS
*)
PLKEY(PICK_LIST, IRC);
MALNO(PICK_LIST, PICK_COUNT, IRC);

IF (PICK_COUNT > 0) AND (?ICK_COUNT <= NUMPK)
THEN
BEGIN
(*
    RETRIEVE THE PICK ENTITY AND PICKED ENTITY
*)
MALGTK(PICK_LIST, NUMPK, PIC_ENTITY, IRC);
MALGTK(PIC_KEY, 1, PICKED_KEY, IRC);
END
ELSE
    IRC  := 18;
END;(* END OF PLQRY *)

F.9
RMBLK SAMPLE PROGRAM

RMBLK - Sample program to implement a delayed "DELETE" operation. It can be reversed by invoking the "REJECT" function.

CA  DESCRIPTION OF ARGUMENTS
  C  INPUT
  C  KEYE  - KEY OF THE ENTITY TO BE DELETED
  C  OUTPUT
  C  IRC  - RETURN CODE
  C  = 0, NORMAL RETURN
  C  = 18, INVALID INPUT DATA

VAR
  NUM_OF_CNSTL : INTEGER;
  NUM_OF_USERS : INTEGER;
  COUNTER : INTEGER;
  CNSTL : LISTKEY;
  POINT_USERS : LISTKEY;
  ENT_HEADER : ENTHEAD;
  DEL_DISPLAY : BOOLEAN;
  ENTITY : ENTBLOCK;

BEGIN (* RMBLK STARTS HERE *)
(*
ADD THE ENTITY TO BE DELETED TO THE ACCEPT LIST
*)
  VCDEL(KEYE, IRC);
(*
DELETE THE DISPLAY OF THE ENTITY
*)
  DELEDSP(KEYE);
(*
RETRIEVE THE ENTITY TO BE DELETED
*)
  MAEGTK(KEYE, ENTITY, IRC);
(*
IF PLURAL ENTITY THEN FIND THE CNSTL
AND OMIT THE DISPLAY OF CNSTL
*)
  IF (ENTITY.PNT.HEADER.KIND = PDDI Access Software_PCPATCH) OR ((ENTITY.PNT.HEADER.KIND = PDDI Access Software_GROUP) OR (ENTITY.PNT.HEADER.KIND = PDDI Access Software_BOUNDED_PLANE))
RMBLK SAMPLE PROGRAM (CONTINUED)

THEN
BEGIN
MAECI(KEYE, CNSTL, IRC);
MALNO(CNSTL, NUM OF CNSTL, IRC);
FOR COUNTER := 1 TO NUM OF CNSTL DO
BEGIN
DEL_DISPLAY := TRUE;
MALGTK(CNSTL, COUNTER, ENT_HEADER, IRC);
(*
IF CONSTITUENT IS A POINT THEN IF THERE
WERE USERS THEN LEAVE IT ALONE
*)
IF ENT HEADER KIND = PDDI Access Software POINT
THEN
BEGIN
MAEU(END HEADER KEY, POINT USERS, IRC);
MALNO(POINT USERS, NUM OF USERS, IRC);
IF NUM OF USERS > 0
THEN
DEL DISPLAY := FALSE;
END;(* ENDIF *)
IF DEL DISPLAY
THEN
DELDSP(ENT HEADER KEY);
END;(* END OF DO LOOP *)
END;(* ENDIF *)
END;(* END OF RMBLK *)
VCASAV SAMPLE PROGRAM

VCASAV - Sample program provides for rejecting changes to attributes.

CA DESCRIPTION OF ARGUMENTS
C INPUT
C OLDENT - OLD ENTITY BLOCK
C
C OUTPUT
C IRC - RETURN CODE
C = 0, NORMAL RETURN
C = 18, INVALID INPUT DATA
*)

VAR
OPERATION_ENTKEY : ENTKEY;
KEYE : ENTKEY;
REJECT_LIST, CNSTL: LISTKEY;
NEWENT : ENTBLOCK;

BEGIN (* VCASAV STARTS HERE *)
(*
GET THE ATTRIBUTE OF THE OLD ENTITY AND
CREATE A NEW ENTITY WITH THAT ATTRIBUTE ONLY
*)
NEWENT := OLDENT;
NEWENT.RPA.UDB.KIND := 0;
CNSTL := 0;
MAECR(KEYE, NEWENT, CNSTL, IRC);
(*
CREATE AN EMPTY LIST AND ADD THE NEWENT TO IT
*)
MAL(CNSTL, IRC);
MALATC(CNSTL, KEYE, IRC);
(*
MODEL THE "REPLACE ATTRIBUTE" OPERATION ENTITY
*)
MODOP(ORD(REPLACE_ATTRIBUTE_OP_), CNSTL, OPERATION_ENTKEY, IRC);
(*
RETRIEVE THE REJECT LIST AND ADD THE NEWLY CREATED OPERATION ENTITY TO THE REJECT LIST
*)
VCRKEY(REJECT_LIST, IRC);
MALATC(REJECT_LIST, OPERATION_ENTKEY, IRC);
END;(* END OF VCASAV *)
KNDRNG SAMPLE PROGRAM

KNDRNG - Sample program checks if the KIND is between a high and a low boundary.

PROCEDURE KNDRNG:

REF

KNDRGI : ROUTINE;

VAR

DATA : BLKDATA;
JRC : INTEGER;

BEGIN (* KNDRNG *)
IF KEYL <> 0 THEN BEGIN
DATA.LOWKIND := LOWKIND;
DATA.HIGHKIND := HIGHKIND;
MALXEQ( KEYL, DATA. KNDRGI, OUTLIST, JRC, IRC );
END ELSE
IRC := 1;
END (* KNDRNG *)

PROCEDURE KNDRGI( CONST KEYENT : ENTKEY;
VAR ENDBLK : ENTBLOCK;
VAR DATA : BLKDATA;
VAR IRC : INTEGER );

SUBPROGRAM

PROCEDURE KNDRGI

BEGIN (* KNDRGI * )
IF ( ENDBLK.KIND >= DATA.LOWKIND )
AND ( ENDBLK.KIND <= DATA.HIGHKIND ) THEN
IRC := 1 (* PUT IT ON THE OUTPUT LIST * )
ELSE
IRC := 2; (* DON'T PUT IT ON THE OUTPUT LIST * )
END; (* KNDRGI * )
SAMPLE PROGRAM

R$RCRD - Sample program retrieves the radius of a PRS. This sample program is written in FORTRAN.

```fortran
FUNCTION R$RCRD(ICV)
  REAL*4 R$RCRD
  REAL*8 RADIUS
  INTEGER*4 IRC,IIRC,I,ICV
  COMMON /PRSRAD/PRSRAD
  CALL MAEXEQ(ICV, RADIUS, PRSRAD, IRC, IIRC)
  R$RCRD = RADIUS
    RETURN
END

PROCEDURE PRSRAD:
BEGIN
  RADIUS := ENTDATA.PRS.RADIUS
END
```

F-14