

LOAN DOCUMENT

PHOTOGRAPH THIS SHEET

DTIC FILE COPY

LEVEL

INVENTORY

AD-A228 379

DTIC ACCESSION NUMBER

WRDC-TR-90-80-27 Vol. IV

DOCUMENT IDENTIFICATION

NOV 1990

DISTRIBUTION STATEMENT A

Approved for public release;
Distribution Unlimited

DISTRIBUTION STATEMENT

ACCESSION FOR	
NTIS	GRA&I
DTIC	TRAC
UNANNOUNCED	
JUSTIFICATION	
BY	
DISTRIBUTION/	
AVAILABILITY CODES	
DISTRIBUTION	AVAILABILITY AND/OR SPECIAL
A-1	

DISTRIBUTION STAMP



DTIC
ELECTE
NOV 02 1990
E D

DATE ACCESSIONED

DATE RETURNED

REGISTERED OR CERTIFIED NUMBER

DATE RECEIVED IN DTIC

PHOTOGRAPH THIS SHEET AND RETURN TO DTIC-FDAC

H
A
N
D
L
E

W
I
T
H

C
A
R
E

AD-A228 379

WRDC-TR-90-8027
Volume IV



GEOMETRIC MODELING APPLICATIONS INTERFACE PROGRAM

GMAP/PDDI SYSTEM COMPONENT PRODUCT SPECIFICATION (AS BUILT)

VOL. IV - System Translator Listings

United Technologies Corporation
Pratt and Whitney
Government Products Division
P.O. Box 9600
West Palm Beach, Florida 33410-9600

November 1990

Final Report For Period August 1985 - March 1989

Approved for public release; distribution is unlimited

MANUFACTURING TECHNOLOGY DIRECTORATE
WRIGHT RESEARCH AND DEVELOPMENT CENTER
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433-6533

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.

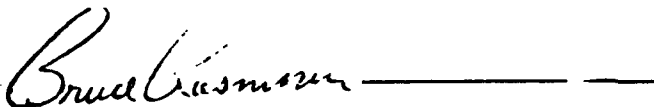
This technical report has been reviewed and is approved for publication.



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 WRDC/MTI, 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.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified			1b. RESTRICTIVE MARKINGS	
2a. SECURITY CLASSIFICATION AUTHORITY			3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited	
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE				
4. PERFORMING ORGANIZATION REPORT NUMBER(S) FR 20889			5. MONITORING ORGANIZATION REPORT NUMBER(S) WRDC-TR-90-8027, Vol. IV	
6a. NAME OF PERFORMING ORGANIZATION United Technologies Corporation Pratt & Whitney Government Products Division		6b. OFFICE SYMBOL (If applicable) (P&W)	7a. NAME OF MONITORING ORGANIZATION Wright Research and Development Center Manufacturing Technology Directorate (WRDC/MTI)	
6c. ADDRESS (City, State and ZIP Code) P.O. Box 9600 West Palm Beach, Florida 33410-9600			7b. ADDRESS (City, State and ZIP Code) Wright-Patterson Air Force Base, OH 45433-6533	
8a. NAME OF FUNDING/SPONSORING ORGANIZATION		8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F33615-85-C-5122	
8c. ADDRESS (City, State and ZIP Code)			10. SOURCE OF FUNDING NOS.	
			PROGRAM ELEMENT NO.	PROJECT NO.
			78011F	MTPI
11. TITLE (Include Security Classification) GEOMETRIC MODELING APPLICATIONS INTERFACE PROGRAM (GMAP)			TASK NO.	WORK UNIT NO.
			06	84
12. PERSONAL AUTHOR(S) R. Disa, C. Van Wie, K. Arnold, J. Altemueller, A. Whelan, G. White, J. Purses				
13a. TYPE OF REPORT Final		13b. TIME COVERED FROM 1 Aug 85 TO 31 MAR 89	14. DATE OF REPORT (Yr., Mo., Day) November 1990	
15. PAGE COUNT 48				
16. SUPPLEMENTARY NOTATION				
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)	
FIELD	GROUP	SUB. GR.	Geometric Modeling Applications Interface Program	
			Product Definition Data Interface	
			Turbine Blades and Disks	
19. ABSTRACT (Continue on reverse if necessary and identify by block number)				
This "As-Built" Product Specification establishes the "as-built" Computer Program Configuration Item (CPCI) identified as the GMAP/PDDI System Components under U.S. Air Force Contract F33615-85-C-5122. It includes descriptions of the structure, functions, language, database requirements, interfaces, and quality assurance provisions of the primary GMAP system components: the System Translator, Model Access Software with Name/Value Interface, and the Schema Manager.				
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐			21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL David Judson			22b. TELEPHONE NUMBER (Include Area Code) (513) 255-7371	22c. OFFICE SYMBOL WRDC/MTI

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 As-built Product Specification, divided into four volumes, covers work performed under Air Force Contract F33615-85-C-5122, Geometric Modeling Applications Interface Program (GMAP), covering the period 1 August 1985 to 31 March 1989. The document addresses the GMAP/PDDI System Components developed or enhanced 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 was the subcontractor responsible for the PDDI System Component work. Mr. Jerry Weiss is the GMAP Program Manager at McDonnell Aircraft and Mr. Herb Ryan is the Deputy Program Manager.

Volume IV of this Product Specification provides the System Translator routine listings and Quality Assurance provisions for the GMAP system components.

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 CM Life Cycle Documentation requirements for a Configuration Item (CI).

TABLE OF CONTENTS

Volume I

		<u>Page</u>
SECTION 1.	SCOPE.....	1-1
1.1	Identification.....	1-1
1.2	Functional Summary.....	1-1
1.3	Approach.....	1-2
SECTION 2.	REFERENCES.....	2-1
2.1	Reference Documents.....	2-1
2.1.1	Military.....	2-1
2.1.2	Commercial.....	2-4
2.1.3	Standards Organizations.....	2-5
2.2	Terms and Acronyms.....	2-6
2.2.1	Glossary A -- Terms Used In GMAP.....	2-6
2.2.2	Glossary B -- Terms Used In IDEF0 Diagrams....	2-19
2.2.3	Acronyms Used In GMAP.....	2-24
SECTION 3.	DETAIL DESIGN.....	3-1
3.1	System Overview.....	3-1
3.1.1	Physical Schemas.....	3-1
3.1.2	Software Packages.....	3-1
3.2	IDEF0 Function Models.....	3-1
3.2.1	Schema Manager.....	3-3
3.2.1.1	A-0 Manage Schema Data.....	3-3
3.2.1.2	A0 - Manage Schema Data.....	3-3
3.2.2	Model Access Software.....	3-7
3.2.2.1	A-0 Perform Model Access.....	3-7
3.2.2.2	A0 Perform Model Access.....	3-7
3.2.3	System Translator.....	3-10
3.2.3.1	A0 Exchange PDD.....	3-10
3.2.3.2	A1 Preprocess PDD.....	3-10
3.2.3.3	A12 Create Data Section.....	3-13
3.2.3.4	A2 Postprocess PDD.....	3-13
3.2.3.5	A23 Process Data Section.....	3-16
3.2.3.6	A24 Resolve Forward References.....	3-16
3.3	Application Interfaces.....	3-19
3.3.1	Interfaces Between GMAP/PDDI	
	System Components.....	3-19
3.3.1.1	Application Program/Working Form.....	3-20
3.3.1.2	User Interface/System Translator.....	3-20
3.3.1.3	Working Form/Exchange Format.....	3-22
3.3.1.4	Working Form/Database	
	Management System.....	3-22

TABLE OF CONTENTS (Continued)

	<u>Page</u>
3.3.1.5 Exchange Format.....	3-22
3.3.1.6 Schema Manager.....	3-23
3.3.2 Interfaces Between Applications.....	3-23
3.3.2.1 Working Form as Exchange Mechanism.....	3-24
3.3.2.2 Working Form as the Native Form.....	3-24
3.3.2.3 Exchange Format and Working Form.....	3-24
3.3.2.4 Transfer Using the Working Form, Exchange Format, and Direct Translator to the Exchange Format.....	3-28
3.4 Program Interrupts.....	3-29
3.4.1 Schema Manager.....	3-29
3.4.2 Model Access Software with Name Value Interface.....	3-31
3.5 Timing and Sequencing Description.....	3-33
3.6 Data Dictionary.....	3-34
3.6.1 Schema Manager.....	3-34
3.6.2 Model Access Software Data Dictionary.....	3-64
3.6.3 Name Value Interface Data Dictionary.....	3-92
3.6.4 System Translator Data Dictionary.....	3-104
3.7 Object Code Creation.....	3-107
3.7.1 Schema Manager.....	3-107
3.7.2 Model Access Software.....	3-107
3.7.3 System Translator.....	3-107
3.7.3.1 Use Portable Higher Order Language.....	3-107
3.7.3.2 Use Model Access Software.....	3-107
3.7.3.3 Interface to Exchange Format.....	3-108
3.7.3.4 Interface to Native System.....	3-108
3.8 Adaptation Data.....	3-108
3.9 Detail Design Description.....	3-108
3.9.1 Schema Manager Hierarchy.....	3-109
3.9.2 Model Access Software Hierarchy.....	3-120
3.9.3 Name/Value Interface Hierarchy.....	3-162
3.9.4 System Translator.....	3-168
3.9.4.1 Postprocessor.....	3-168
3.9.4.2 Preprocessor.....	3-170

TABLE OF CONTENTS (Continued)

		<u>Page</u>
Volume II		
3.10	Routine Listings.....	3-171
3.10.1	Schema Manager.....	3-171
3.10.1.1	Index.....	3-171
3.10.1.2	Listings.....	3-178
Volume III		
3.10.2	Model Access Software.....	3-625
3.10.2.1	Index.....	3-625
3.10.2.2	Listings.....	3-631
3.10.3	N/VI.....	3-913
3.10.3.1	Index.....	3-913
3.10.3.2	Listings.....	3-914
Volume IV		
3.10.4	System Translator.....	3-979
3.10.4.1	Index.....	3-979
3.10.4.2	Listings.....	3-980
SECTION 4	QUALITY ASSURANCE.....	4-1
4.1	Quality Assurance (QA) Requirements.....	4-1

3.10.4 System Translator

3.10.4.1 Index

<u>Routine</u>	<u>Function</u>
CHECK	- Checks the processed flag on an entity
CRHEAD	- Creates the header section of the STEP exchange format file
GETID	- Retrieves the entity identifier associated with an entity key
GETKIND	- Determines the entity kind number from the entity name
GETTOK	- Will parse the exchange format file based on the value of tokentype
GETVAL	- Extracts a character string from input file buffer, converts it to it's internal format, stores it directly to the entity's ADB
GTKEY	- Retrieves the entity key associated with an entity identifier
LTRIM	- Takes a varying character string and removes any leading blanks
NEWDD	- Reads in data dictionary descriptions of entities based upon their kind number
POST	- Initializes the translator environment and directs the translation process
PRE	- Extracts entities from working form model and places them into a STEP exchange format file
PRHEAD	- Reads the header section of the STEP exchange format file
PUTID	- Saves the entity identifier associated with an entity key
PUTKEY	- Stores the entity key associated with an entity identifier
RDENT	- Reads the exchange format entity into the working form model
RESOLVE	- Checks for unresolved forward references and prints diagnostic information about them
RTRIM	- Takes a varying character string and removes any leading blanks
WRTENT	- Writes the mas entity out to the STEP exchange format file

3.10.4.2 Listings

MODULE CHECK;

```
(*-----*)
(*)
(*) Author   : Phil Dorr           Created: 01-MAR-1988 (*)
(*) Version : 1.0                 Revised:          (*)
(*)
(*) Routine : Check               (*)
(*)
(*) Function:                     (*)
(*)   This routine will check the processed flag on an entity (*)
(*)   and if it is not set it will call WRTENT to process the (*)
(*)   entity. This routine is only called from an 'XEQ' routine. (*)
(*)
(*) Environment:                  (*)
(*)   VAX Pascal V3.3 and VMS 4.4 (*)
(*)
(*) Called by:                    (*)
(*)   PRE      (via MAKXEQ)        (*)
(*)   WRTENT   (via MAECXQ)        (*)
(*)
(*) Calls:                        (*)
(*)   WRTENT (*)
(*)
(*)-----*)

(*--- Restrictions ---*)
(*)
(*) No known restrictions (4 unknown restrictions) (*)
(*)
(*)-----*)

(*--- Commons ---*)
(*)
(*) No commons are used. (*)
(*)
(*)-----*)

(*--- Processing Description ---*)
(*)
(*) Call MAS routine MAESVL to determine if the entities pro- (*)
(*) cessed flag is set. (*)
(*) If the flag is not set then call WRTENT to process entity. (*)
(*)
(*)--- Data Structures/Major Variables ---*)
(*)
```

CI PS560240032U
April 1990

(* FLAG The value of the processed flag of the current *)
(* entity. *)
(* *)
(*-----*)

(*-- Change Control -----*)
(* *)
(* Changed by: Date: *)
(* *)
(* *)
(*-----*)

MODULE CRHEAD;

```
(*-----*)
(*)
(*) Author : J.Altemueller          Created: 3/1/88      (*)
(*) Version : 1.0                    Revised:           (*)
(*)
(*) Routine : CRHEAD                (*)
(*)
(*) Function:                        (*)
(*)   This routine is responsible for creating the header section (*)
(*)   of the STEP exchange format file                          (*)
(*)
(*) Environment:                     (*)
(*)   VAX Pascal V3.3 and VMS 4.4                                  (*)
(*)
(*) Called by:                       (*)
(*)   PRE                                                            (*)
(*)
(*) Calls:                           (*)
(*)   No other routines                                             (*)
(*)-----*)

(*--- Restrictions -----*)
(*)           None                                                  (*)
(*---*)

(*--- Files -----*)
(*)   This routine accesses the file "PASFIL" for read purposes only (*)
(*)   This routine accesses the file "EFFILE" for write purposes only (*)
(*---*)

(*--- Commons -----*)
(*)           None                                                  (*)
(*---*)

(*--- Processing Description -----*)
(*)   This routine will read in the file identification data from   (*)
(*)   PASFIL, reformat it in the form required by STEP, and then   (*)
(*)   write it to the the exchange format file (EFFILE)             (*)
(*)
(*)-----*)

(*--- Data Structures/Major Variables -----*)
(*)           None                                                  (*)
(*---*)
```

CI PS560240032U
April 1990

(*--- Change Control -----*)
(* *)
(* Changed by: Date: *)
(* *)
(* *)
(*-----*)

MODULE GETID;

```
(*-----*)
(*)
(*) Author : P.D.Dorr Created: 3/1/88 (*)
(*) Version : 1.0 Revised: (*)
(*)
(*) Routine : GETID (*)
(*)
(*) Function: (*)
(*) Retrieve the entity identifier associated with an entity key (*)
(*)
(*) Environment: (*)
(*) VAX Pascal V3.3 and VMS 4.4 (*)
(*)
(*) Called by: (*)
(*) WRTENT (*)
(*)
(*) Calls: (*)
(*) None (*)
(*)
(*)-----*)

(*--- Restrictions ---*)
(*) None (*)
(*)
(*)-----*)

(*--- Files ---*)
(*)
(*) This routine accesses no external files (*)
(*)
(*)
(*)-----*)

(*--- Commons ---*)
(*) See the PRECOM include file documentation (*)
(*)
(*)-----*)

(*--- Processing Description ---*)
(*) This routine will retrieve the entity identifier from the (*)
(*) IDENT field of the entity's attribute data block (*)
(*)
(*)-----*)
```

```
(*--- Data Structures/Major Variables -----*)  
(*                                                                                      *)  
(* input variable MASKEY : entkey of the mas entity of interest *)  
(* input variable ADB    : attribute data block of the mas entity *)  
(* output variable CHARPTR : entity identifier of the STEP entity *)  
(* output variable  RC    : return code 0 => success *)  
(*                                                                                      *)  
(*-----*)
```

```
(*--- Change Control -----*)  
(*                                                                                      *)  
(*      Changed by:                               Date: *)  
(*                                                                                      *)  
(*                                                                                      *)  
(*-----*)
```

MODULE GETKIND;

```
(*-----*)
(*)
(*) Author : P.Dorr Created: 4/25/88 (*)
(*) Version : 1.0 Revised: (*)
(*)
(*) Routine : getkind (*)
(*)
(*) Function: (*)
(*) This routine will determine the entity kind number from the (*)
(*) entity name. (*)
(*)
(*) Environment: (*)
(*) VAX Pascal V3.3 and VMS 4.4 (*)
(*)
(*) Called by: (*)
(*) RDENT (*)
(*)
(*) Calls: (*)
(*) None (*)
(*)
(*)-----*)

(*--- Restrictions -----*)
(*) None (*)
(*-----*)

(*--- Files -----*)
(*) This routine accesses no files. (*)
(*-----*)

(*--- Commons -----*)
(*) See the POSTCOM include file documentation. (*)
(*) See the V5INCLD include file documentation. (*)
(*-----*)

(*--- Processing Description -----*)
(*) This routine will access the data dictionary index area (*)
(*) searching for a match to the entity name passed into this (*)
(*) routine. When a match is found, the data dictionary will also (*)
(*) contain the corresponding entity kind number. That kind number (*)
(*) is then passed back from this routine. (*)
(*)
(*)-----*)
```

```
(*--- Data Structures/Major Variables -----*)
(*)
(*)   input  NAME   : varying string          (*)
(*)   output KIND  : integer                   (*)
(*)   output RC    : integer                   (*)
(*)
(*)-----*)
```

```
(*--- Change Control -----*)
(*)
(*)   Changed by:                               Date:          (*)
(*)
(*)
(*)-----*)
```

MODULE GETTOK;

```

(*)-----(*)
(*)
(*) Author : Phil Dorr Created: 28-MAR-88 (*)
(*) Version : 1.0 Revised: (*)
(*)
(*) Routine : GETTOK (*)
(*)
(*) Function: (*)
(*) This routine will parse the exchange format file based on (*)
(*) the value of tokentype. (*)
(*)
(*) Environment: (*)
(*) VAX Pascal V3.3 and VMS 4.4 (*)
(*)
(*) Called by: (*)
(*) GTCON (*)
(*) GTVAL (*)
(*) PRHEAD (*)
(*) RDENT (*)
(*)
(*) Calls: (*)
(*) READNEXT (internal routine) - reads next record (*)
(*) SCAN (internal routine) - finds next delimiter (*)
(*)
(*)-----(*)

(*)--- Restrictions -----(*)
(*)
(*) This routine does not handle double apostrophes embedded in (*)
(*) strings when the first apostrophe is at the end of one record (*)
(*) and the other is at the start of another record. (*)
(*)
(*)-----(*)

(*)--- Arguments -----(*)
(*)
(*) TOKENTYPE (input) type = TOKEN_CLASS (*)
(*) One of: ENTITYID_TYP, KEYWORD_TYP, OPENLIST_TYP, (*)
(*) CLOSELIST_TYP, SEMICOLON_TYP, STRING_TYP, or (*)
(*) ENTITYREF_TYP. (*)
(*) This is the type of token that is expected next. (*)
(*)
(*) KEYWORD (output) type = ENT_NAME (*)
(*) This argument is only used for tokens of type KEYWORD_TYP. *)

```

```

(*)
(*) ENTITYID (output) type = INTEGER (*)
(*) This argument is only used for tokens of type ENTITYID_TYP. (*)
(*)
(*) RC (output) type = INTEGER (*)
(*) A return code <0 is an informational message, a return (*)
(*) code =0 is a mission accomplished, and a return code >0 (*)
(*) is an error. See below for meaning of return codes. (*)
(*)
(*)-----(*)

(*)--- Commons -----(*)
(*)
(*)
(*)-----(*)

(*)--- Processing Description -----(*)
(*)
(*)
(*)-----(*)

(*)--- Return Codes -----(*)
(*)
(*) < 0 I gives delimiter type (negated) that followed (*)
(*) 0 I required delimiter found (*)
(*) 1 E required delimiter not found (*)
(*) 2 E invalid format for a value or file syntax error (*)
(*) 3 E invalid calling args (programmer error) (*)
(*) 4 S file read error (maybe early end-of-file) (*)
(*) 5 I possible embedded entity (*)
(*) 6 I defaulted entity (*)
(*) 7 E data skipped (*)
(*) 8 I end of list (*)
(*)
(*)
(*) ,-' '- I=Informational, E = Error, S= Severe error (*)
(*) '----- Return code value (*)
(*)
(*) A return code of 4 is most likely terminal, the others may (*)
(*) be recovered. (*)
(*)
(*)-----(*)

(*)--- Delimiter Type -----(*)
(*)
(*) 1 = COMMA, (*)
(*) 2 = OPEN_PARENTHESIS (*)

```

April 1990

```

(*)      3 = CLOSE_PARENTHESIS                      *)
(*)      4 = SEMICOLON                               *)
(*)      5 = EQUAL_SIGN                              *)
(*)      6 = AT_SIGN                                 *)
(*)      7 = POUND_SIGN                              *)
(*)      8 = EXCLAMATION_POINT                      *)
(*)      9 = APOSTROPHE                             *)
(*)
(*)-----*)
(*)--- Data Structures/Major Variables -----*)
(*)
(*)      BUFFER_POS  the current position into the exchange format *)
(*)                  file record buffer.                        *)
(*)      BUILDSTR   in the event a lexical element spans several records *)
(*)                  BUILDSTR will contain the entire element. See *)
(*)                  comments on USE_STR.                        *)
(*)      DELIMS      a string that contains all the delimiters that the *)
(*)                  SCAN recognizes. The delimiter number is the *)
(*)                  position in the DELIMS string of the delimiter. *)
(*)      DELIM_POS   the position of the next delimiter. DELIM_POS is *)
(*)                  always >= BUFFER_POS.                      *)
(*)      DYTPE       the delimiter number of the delimiter that SCAN *)
(*)                  has most recently found. (1 <= DTYPE <= 8 ). *)
(*)      USE_STR     if this boolean is TRUE then the current lexical *)
(*)                  element spans 2 or more records. If it is false then *)
(*)                  DELIM_POS is valid and should used to get at element *)
(*)                  in record buffer.                          *)
(*)
(*)-----*)

(*)--- Change Control -----*)
(*)
(*)      Changed by:                               Date:      *)
(*)
(*)-----*)

```

MODULE GETVAL;

```
(*-----*)
(*)
(*) Author : P. Dorr Created: 10-MAY-1988 (*)
(*) Version : 1.0 Revised: (*)
(*)
(*) Routine : GETVAL (*)
(*)
(*) Function: (*)
(*) This routine will extract a character string from the input (*)
(*) file buffer and convert it to it's internal format and store (*)
(*) it directly to the entity's ADB. (*)
(*)
(*) Environment: (*)
(*) VAX Pascal V3.3 and VMS 4.4 (*)
(*)
(*) Called by: (*)
(*) MAKEENT (in module RDENT) via MAEXEQ (*)
(*)
(*) Calls: (*)
(*) READNEXT (internal routine) - reads next record (*)
(*) SCAN (internal routine) - finds next delimiter (*)
(*)
(*)-----*)
```

```
(*--- Restrictions -----*)
(*)
(*) This routine does not detect the difference between integers (*)
(*) and real numbers when reading into a real number in an ADB. (*)
(*)
(*) This routine will also accept a syntax for a real number that (*)
(*) has no digits to the right of the decimal point. (*)
(*)
(*) This routine does not handle double apostrophes embedded in (*)
(*) strings when the first apostrophe is at the end of one record (*)
(*) and the other is at the start of another record. (*)
(*)
(*)-----*)
```

```
(*--- Commons -----*)
(*)
(*)
(*)-----*)
```

```
(*--- Processing Description -----*)
```

```
(*)
(*) scan to the next delimiter, call readnext if nessesary (*)
(*) readv into the ADB (based on the type of number of course) (*)
(*)
(*)-----(*)
(*)--- Return Codes -----(*)
(*)
(*) < 0 I gives delimiter type (negated) that followed (*)
(*) 0 I required delimiter found (*)
(*) 1 E required delimiter not found (*)
(*) 2 E invalid format for a value or file syntax error (*)
(*) 3 E invalid calling args (programmer error) (*)
(*) 4 S file read error (maybe early end-of-file) (*)
(*) 5 I possible embedded entity (*)
(*) 6 I defaulted entity (*)
(*) 7 E data skipped (*)
(*) 8 I end of list (*)
(*)
(*)
(*) ,-' '- I=Informational, E = Error, S= Severe error (*)
(*) '----- Return code value (*)
(*)
(*) A return code of 4 is most likely terminal, the others may (*)
(*) be recovered. (*)
(*)
(*)-----(*)

(*)--- Delimiter Type -----(*)
(*)
(*) 1 = COMMA, (*)
(*) 2 = OPEN_PARENTHESIS (*)
(*) 3 = CLOSE_PARENTHESIS (*)
(*) 4 = SEMICOLON (*)
(*) 5 = EQUAL_SIGN (*)
(*) 6 = AT_SIGN (*)
(*) 7 = POUND_SIGN (*)
(*) 8 = EXCLAMATION_POINT (*)
(*) 9 = APOSTROPHE (*)
(*)
(*)-----(*)

(*)--- Data Structures/Major Variables -----(*)
(*)
(*) BUFFER_POS the current position into the exchange format (*)
(*) file record buffer. (*)
(*) BUILDSTR in the event a lexical element spans several records (*)
(*) BUILDSTR will contain the entire element. See (*)
```

April 1990

```

(*)      comments on USE_STR.                                *)
(*)  DELIMS  a string that contains all the delimiters that the *)
(*)          SCAN recognizes. The delimiter number is the      *)
(*)          position in the DELIMS string of the delimiter.    *)
(*)  DELIM_POS the position of the next delimiter. DELIM_POS is *)
(*)          always >= BUFFER_POS.                              *)
(*)  DYTPE   the delimiter number of the delimiter that SCAN   *)
(*)          has most recently found. (1 <= DTYPE <= 8 ).      *)
(*)  USE_STR  if this boolean is TRUE then the current lexical  *)
(*)          element spans 2 or more records. If it is false then *)
(*)          DELIM_POS is valid and should used to get at element *)
(*)          in record buffer.                                  *)
(*)-----*)

(*)--- Change Control -----*)
(*)                                                                    *)
(*)  Changed by:                               Date:                *)
(*)                                                                    *)
(*)-----*)

```

MODULE GTKEY;

```
(*-----*)
(*)
(*) Author : P.Dorr Created: 4/25/88 (*)
(*) Version : 1.0 Revised: (*)
(*) (*)
(*) Routine : GTKEY (*)
(*) (*)
(*) Function: (*)
(*) This routine will retrieve the entity key associated with an (*)
(*) entity identifier. (*)
(*) (*)
(*) Environment: (*)
(*) VAX Pascal V3.3 and VMS 4.4 (*)
(*) (*)
(*) Called by: (*)
(*) RDENT (*)
(*) (*)
(*) Calls: (*)
(*) maecrn (*)
(*) (*)
(*)-----*)

(*--- Restrictions ---*)
(*) None (*)
(*) (*)
(*)-----*)

(*--- Files ---*)
(*) This routine uses no external files (*)
(*) (*)
(*)-----*)

(*--- Commons ---*)
(*) See the POSTCOM include file documentation (*)
(*) See the VXINCLD include file documentation (*)
(*) (*)
(*)-----*)

(*--- Processing Description ---*)
(*) (*)
(*) If the entity identifier and maskey are both valid entries (*)
(*) then the maskey associated with the entity identifier is (*)
(*) retrieved from a static table (KEYTABLE). If the entity (*)
(*) identifier is a forward reference, then a forward reference (*)
(*) entity is created and its maskey is returned. (*)
(*) (*)
```

```
(*-----*)
(*-- Data Structures/Major Variables -----*)
(*      Input  ENTITY_ID : Integer      *)
(*      Output MASKEY   : Entkey        *)
(*      Output RC       : Integer        *)
(*                                     *)
(*-----*)

(*-- Change Control -----*)
(*                                     *)
(*      Changed by:                      Date:      *)
(*                                     *)
(*-----*)
```

MODULE LTRIM;

```
(*-----*)
(*)
(*)  Author   : Phil Dorr           Created: 02-MAR-1988  (*)
(*)  Version : 1.0                 Revised:           (*)
(*)
(*)  Routine  : LTRIM              (*)
(*)
(*)  Function:
(*)    This routine will take a varying character string and
(*)    remove any leading blanks.
(*)
(*)  Environment:
(*)    VAX Pascal V3.3 and VMS 4.4
(*)
(*)  Called by:
(*)    WRTENT
(*)
(*)  Calls:
(*)    (none)
(*)
(*)-----*)

(*--- Restrictions -----*)
(*)
(*)  VAX Pascal can only handle strings with lengths less than
(*)  65,535 characters.
(*)
(*)-----*)

(*--- Commons -----*)
(*)
(*)  No commons used.
(*)
(*)-----*)

(*--- Processing Description -----*)
(*)
(*)  Scan through the string looking for the first nonblank character
(*)  Assign the substring of the input string, starting at the
(*)  first nonblank character and running to the end of the
(*)  string, to the output string.
(*)
(*)-----*)

(*--- Data Structures/Major Variables -----*)
(*)
(*)  LEN      The length of the STR when LTRIM is called.
(*)
```

(* POS The position in STR where the first noblank character *)
(* occurs. *)
(* STR The string to be trimmed. This is passed to LTRIM by *)
(* reference. *)
(*-----*)

(*--- Change Control -----*)
(* *)
(* Changed by: Date: *)
(* *)
(* *)
(*-----*)

MODULE NEWDD;

```
(*-----*)
(*)
(*) Author : P.D.Dorr Created: 3/1/88 (*)
(*) Version : 1.0 Revised: (*)
(*)
(*) Routine : NEWDD (*)
(*)
(*) Function: (*)
(*) The function of this routine is to read in data dictionary (*)
(*) descriptions of entities based upon their kind number. (*)
(*)
(*) Environment: (*)
(*) VAX Pascal V3.3 and VMS 4.4 (*)
(*)
(*) Called by: (*)
(*) WRTEXT (*)
(*) RDENT (*)
(*) Calls: (*)
(*) None (*)
(*)-----*)

(*--- Restrictions -----*)
(*) This routine will store up to TABLE_SIZE number of (*)
(*) distinct data dictionary entries. TABLE_SIZE is a predefined (*)
(*) constant that can be found in the PRECOM include file (*)
(*)-----*)

(*--- Files -----*)
(*)
(*) This routine will access the logical file DDFILE for reading (*)
(*)-----*)

(*--- Commons -----*)
(*) None (*)
(*)-----*)

(*--- Processing Description -----*)
(*)
(*) Using information from the data dictionary index file, this (*)
(*) routine will read in a data dictionary entry from the DDFILE (*)
(*) that corresponds to a given entity kind number. the data that (*)
(*) is pertinent to the translator is converted to integer and (*)
```

```
(*  stored into a common queue. If the queue is full when a request *)
(*  is made, then the oldest entry on the queue is replaced.      *)
(*  If an entry of the correct entity kind already exists in the  *)
(*  queue then a new data dictionary entry is not read from DDFILE *)
(*  -----*)
```

```
(*--- Data Structures/Major Variables -----*)
(*  See the PRECOM include file                                     *)
(*  -----*)
```

```
(*--- Change Control -----*)
(*  Changed by:                                                    Date: *)
(*  -----*)
```

MODULE POST;

```
(*-----*)
(*)
(*) Author : J.Altemueller Created: 3/22/88 (*)
(*) Version : 1.0 Revised: (*)
(*)
(*) Routine : POST (*)
(*)
(*) Function: (*)
(*) This routine is responsible for initializing the translator (*)
(*) environment and directing the translation process (*)
(*)
(*) Environment: (*)
(*) VAX Pascal V3.3 and VMS 4.4 (*)
(*)
(*) Called by: (*)
(*)
(*) Calls: (*)
(*) mainit,makxreq,maecrn (*)
(*) rdindx (*)
(*) prhead (*)
(*) rdent (*)
(*)-----*)

(*--- Restrictions -----*)
(*) None (*)
(*)
(*)-----*)

(*--- Files -----*)
(*)
(*) This routine will allocate the following files (*)
(*) POSTEF = the exchange format file (*)
(*) DDINX = data dictionary index file (*)
(*) DDFILE = data dictionary file (*)
(*) ERRMSG = error messages file (*)
(*)
(*)-----*)

(*--- Commons -----*)
(*)
(*)
(*)
(*)-----*)
```

```
(*--- Processing Description -----*)
(*)
(*)      open files                      (*)
(*)      initialize common variables    (*)
(*)      Setup MAS enviroment           (*)
(*)      Read in the data dictionary index file (*)
(*)      process HEADER section of exchange format file (*)
(*)      process DATA section of exchange format file (*)
(*)      close files                    (*)
(*)
(*)-----*)

(*--- Data Structures/Major Variables -----*)
(*)
(*)      See the POSTCOM include file documentation (*)
(*)      See the V%INCLD include file documentation (*)
(*)
(*)-----*)

(*--- Change Control -----*)
(*)
(*)      Changed by:                      Date: (*)
(*)
(*)
(*)-----*)
```

PROGRAM PRE(EFFILE, DDFILE, DDINX, PASFIL, DATA);

```
(*-----*)
(*                                          *)
(*  Author   : P.D.Dorr                    Created: 3/1/88  *)
(*  Version  : 1.0                        Revised:          *)
(*                                          *)
(*  Routine  : PRE                        *)
(*                                          *)
(*  Function:                            *)
(*    The function of this routine is to extract entities  *)
(*    from the working form model and place them into a STEP *)
(*    exchange format file                                *)
(*                                          *)
(*  Environment:                            *)
(*    VAX Pascal V3.3 and VMS 4.4          *)
(*                                          *)
(*  Called by:                            *)
(*                                          *)
(*  Calls:                                   *)
(*    RDINX                                *)
(*    maeknd                               *)
(*    maectk                               *)
(*    makxreq                              *)
(*    FILRTV                              *)
(*    CRHEAD                              *)
(*-----*)

(*--- Restrictions ---*)
(*    None *)
(*---*)

(*--- Files ---*)
(*    This routine will open and close the following files: *)
(*    DDFILE...READ ACCESS *)
(*    DDINX...READ ACCESS  *)
(*    EFFILE...WRITE ACCESS *)
(*---*)

(*--- Commons ---*)
(*    See the PRECOM include file documentation *)
(*---*)
```

```
(*--- Processing Description -----*)
(*)
(*)      open files                      (*)
(*)      Read index into internal structure (*)
(*)      initialize common variables      (*)
(*)      retrieve the model from PID       (*)
(*)      begin the step file              (*)
(*)      create the header section        (*)
(*)      begin the data section           (*)
(*)      obtain the number of entity kinds in the model (*)
(*)      process all the entity kinds in the model (*)
(*)      (except array entities kind = 1100 ) (*)
(*)      end the data section and the step file (*)
(*)      close files                      (*)
(*)-----*)

(*--- Data Structures/Major Variables -----*)
(*)      See the PRECOM include file documentation (*)
(*)-----*)

(*--- Change Control -----*)
(*)
(*)      Changed by:                      Date: (*)
(*)-----*)
```

MODULE PRHEAD;

```
(*-----*)
(*)
(*) Author : J.Altemueller Created: 3/1/88 (*)
(*) Version : 1.0 Revised: (*)
(*)
(*) Routine : PRHEAD (*)
(*)
(*) Function: (*)
(*) This routine is responsible for reading the header section (*)
(*) of the STEP exchange format file (*)
(*)
(*) Environment: (*)
(*) VAX Pascal V3.3 and VMS 4.4 (*)
(*)
(*) Called by: (*)
(*) POST (*)
(*)
(*) Calls: (*)
(*) GETTOK (*)
(*)
(*)-----*)

(*--- Restrictions -----*)
(*) None (*)
(*---*)

(*--- Files -----*)
(*) This routine accesses the file "POSTEF" for read purposes only (*)
(*---*)

(*--- Commons -----*)
(*) None (*)
(*---*)

(*--- Processing Description -----*)
(*) This routine will read in the HEADER section from the STEP (*)
(*) exchange format file (POSTEF) (*)
(*)
(*)-----*)

(*--- Data Structures/Major Variables -----*)
(*) None (*)
(*---*)
```

```
(*--- Change Control -----*)
(*)
(*)   Changed by:                               Date:   (*)
(*)
(*)-----*)
%include 'V5INCLD'
type ENTBLOCK = integer;

[global]
procedure PRHEAD( var irc      : integer);

procedure GETTOK( var TOKENTYPE : token_classes;
                  var KEYWORD   : workstr;
                  var ENTITYID  : integer;
                  var RC        : integer );

external;

label 9999;
var   DUMMY      : integer;
      KEYWORD    : workstr;
      TOKENTYP   : token_classes;
      RC         : integer;

begin

    irc := 0;

    (*****)
    (*      read in the "STEP;" keyword      *)
    (*****)
    tokentyp := KEYWORD_TYP;
    gettok( TOKENTYP, KEYWORD, DUMMY, RC );
    if( RC <> -4 )or( KEYWORD <> 'STEP' )then begin
        (* -4 indicated a ";" *)
        IRC := 1;
        goto 9999
    end;

    (*****)
    (*      read in the "HEADER;" keyword      *)
    (*****)
    gettok( TOKENTYP, KEYWORD, DUMMY, RC );
    if( RC <> -4 )or( KEYWORD <> 'HEADER' )then begin
        (* -4 indicated a ";" *)
        IRC := 1;
        goto 9999
    end;
```

```
(*****)
(*      read in the "file_identification" entity      *)
(*****)

(*  file_identification keyword  *)
  gettok( TOKENTYP, KEYWORD, DUMMY, RC );
  if( RC <> -2 )or( KEYWORD <> 'FILE_IDENTIFICATION' )then begin
    (* -2 indicated a "(" *)
    IRC := 1;
    goto 9999
  end;

(*  file_name *)
  tokentyp := STRING_TYP;
  gettok( TOKENTYP, KEYWORD, DUMMY, RC );
  if RC <> -1 then begin
    (* -1 indicated a "," *)
    IRC := 1;
    goto 9999
  end;

(*  date *)
  gettok( TOKENTYP, KEYWORD, DUMMY, RC );
  if RC <> -1 then begin
    (* -1 indicated a "," *)
    IRC := 1;
    goto 9999
  end;

(*  author *)
  gettok( TOKENTYP, KEYWORD, DUMMY, RC );
  if RC <> -1 then begin
    (* -1 indicated a "," *)
    IRC := 1;
    goto 9999
  end;

(*  org *)
  gettok( TOKENTYP, KEYWORD, DUMMY, RC );
  if RC <> -1 then begin
    (* -1 indicated a "," *)
    IRC := 1;
    goto 9999
  end;
```

```
(* step version *)
  gettok( TOKENTYP, KEYWORD, DUMMY, RC );
  if RC <> -1 then begin
    (* -1 indicated a "," *)
    IRC := 1;
    goto 9999
  end;

(* preprocessor version *)
  gettok( TOKENTYP, KEYWORD, DUMMY, RC );
  if RC <> -1 then begin
    (* -1 indicated a "," *)
    IRC := 1;
    goto 9999
  end;

(* originating system *)
  gettok( TOKENTYP, KEYWORD, DUMMY, RC );
  if RC <> -3 then begin
    (* -3 indicated a ")" *)
    IRC := 1;
    goto 9999
  end;

(* closing semi-colon *)
  tokentyp := SEMICOLON_TYP;
  gettok( TOKENTYP, KEYWORD, DUMMY, RC );

(*****)
(*          read in the file description          *)
(*****)

(* file_description keyword *)
  tokentyp := KEYWORD_TYP;
  gettok( TOKENTYP, KEYWORD, DUMMY, RC );
  if( RC <> -2 )or( KEYWORD <> 'FILE_DESCRIPTION' )then begin
    (* -2 indicated a "(" *)
    IRC := 1;
    goto 9999
  end;

(* file_description *)
  tokentyp := string_typ;
  gettok( TOKENTYP, KEYWORD, DUMMY, RC );
```

```
(* closing semi-colon *)
  tokentyp := SEMICOLON_TYP;
  gettok( TOKENTYP, KEYWORD, DUMMY, RC );

(*****)
(*          read in the implementation level          *)
(*****)

(* implementation level keyword *)
  tokentyp := KEYWORD_TYP;
  gettok( TOKENTYP, KEYWORD, DUMMY, RC );
  if( RC <> -2 )or( KEYWORD <> 'FILE_DESCRIPTION' )then begin
    (* -2 indicated a "(" *)
    IRC := 1;
    goto 9999
  end;

(* implementation level *)
  tokentyp := STRING_TYP;
  gettok( TOKENTYP, KEYWORD, DUMMY, RC );

(* closing semi-colon *)
  tokentyp := SEMICOLON_TYP;
  gettok( TOKENTYP, KEYWORD, DUMMY, RC );

(*****)
(*          skip remainder of header section looking for "endsec;"          *)
(*****)

9999:
  tokentyp := KEYWORD_SEARCH;
  keyword := 'ENDSEC';
  gettok( TOKENTYP, KEYWORD, DUMMY, RC );
  if RC <> -4 then begin
    (* -4 indicated a ";" *)
    IRC := 1
  end;

end;
end.
```

MODULE PUTID;

```
(*-----*)
(*)
(*) Author : P.D.Dorr Created: 3/3/88 (*)
(*) Version : 1.0 Revised: (*)
(*)
(*) Routine : PUTID (*)
(*)
(*) Function: (*)
(*) The function of this routine is to save the entity (*)
(*) identifier associated with an entity key (*)
(*)
(*) Environment: (*)
(*) VAX Pascal V3.3 and VMS 4.4 (*)
(*)
(*) Called by: (*)
(*) WRTENT (*)
(*)
(*) Calls: (*)
(*)
(*)-----*)

(*--- Restrictions -----*)
(*) None (*)
(*)
(*)-----*)

(*--- Files -----*)
(*) This routine accesses no external files (*)
(*)
(*)-----*)

(*--- Commons -----*)
(*) See the PRECOM include file documentation (*)
(*)
(*)-----*)

(*--- Processing Description -----*)
(*) This routine will store the address of the entity (*)
(*) identifier string into the IDENT field of the entity identifier (*)
(*)
(*)-----*)
```

```
(*--- Data Structures/Major Variables -----*)
(*)
(*) input variable ADB      : attribute data block      (*)
(*) input variable CHARID : entity identifier          (*)
(*) output variable RC     : return code                (*)
(*)
(*)-----*)

(*--- Change Control -----*)
(*)
(*)   Changed by:                               Date:    (*)
(*)
(*)
(*)-----*)
```

MODULE PUTKEY;

```

(*)-----*)
(*)*)
(*)  Author   : P.Dorr                Created: 4/25/88      *)
(*)  Version  : 1.0                  Revised:              *)
(*)*)
(*)  Routine  : PUTKEY                *)
(*)*)
(*)  Function:                        *)
(*)    This routine will store the entity key associated with an *)
(*)    entity identifier for later reference.                    *)
(*)*)
(*)  Environment:                      *)
(*)    VAX Pascal V3.3 and VMS 4.4                                *)
(*)*)
(*)  Called by:                        *)
(*)    RDENT                                                         *)
(*)*)
(*)  Calls:                             *)
(*)    maeu,malno,malgtk,malfnd,malrpl,maed,mald                 *)
(*)*)
(*)-----*)

(*)--- Restrictions -----*)
(*)  None                                                         *)
(*)*)

(*)--- Files -----*)
(*)  This routine uses no external files                          *)
(*)*)

(*)--- Commons -----*)
(*)  See the POSTCOM include file documentation                  *)
(*)  See the V%INCLD include file documentation                 *)
(*)*)

(*)--- Processing Description -----*)
(*)*)
(*)  If the entity identifier and maskey are both valid entries *)
(*)  then the maskey associated with the entity identifier is stored *)
(*)  into a static table (KEYTABLE). If the entity identifier is *)
(*)  associated with a forward reference, then the new maskey is *)
(*)  replaced in all necessary constituent lists.                *)
(*)*)

```

```
(*--- Data Structures/Major Variables -----*)  
(*      Input  ENTITY_ID : Integer          *)  
(*      Input  MASKEY   : Entkey            *)  
(*      Output RC        : Integer          *)  
(*-----*)
```

```
(*--- Change Control -----*)  
(*-----*)  
(*      Changed by:                      Date:          *)  
(*-----*)  
(*-----*)
```

MODULE RDENT;

```
(*-----*)
(*)
(*) Author : P.D.Dorr Created: 4.20/88 (*)
(*) Version : 1.0 Revised: (*)
(*)
(*) Routine : RDENT (*)
(*)
(*) Function: (*)
(*) The function of this routine is to read the exchange format (*)
(*) entity into the working form model (*)
(*)
(*) Environment: (*)
(*) VAX Pascal V3.3 and VMS 4.4 (*)
(*)
(*) Called by: (*)
(*) POST (*)
(*)
(*) Calls: (*)
(*) GETKIND,GETTOK (*)
(*) NEWDD (*)
(*) GETKEY,PUTKEY (*)
(*) maecrn,maexeq,malatc,malrpl (*)
(*)
(*)-----*)

(*--- Restrictions ---*)
(*) None (*)
(*)
(*)-----*)

(*--- Files ---*)
(*) This routine accesses EFFILE (*)
(*)-----*)

(*--- Commons ---*)
(*) See the POSTCOM include file documentation (*)
(*) See the V5INCLD include file documentation (*)
(*)-----*)

(*--- Processing Description ---*)
(*) As each entity is read, its entity keyword is converted (*)
(*) into the appropriate MAS entity kind value, and a data dictionary*)
(*) is obtained. Then each attribute is read in, and placed into the*)
(*) MAS entity according to the data dictionary description for the *)
(*) attribute. (*)
(*)
```

CI PS560240032U
April 1990

```
(*-----*)  
(*--- Data Structures/Major Variables -----*)  
(*      Output IRC : integer 0 -> success, <> 0  -> failure      *)  
(*-----*)  
  
(*--- Change Control -----*)  
(*-----*)  
(*      Changed by:                               Date:          *)  
(*-----*)  
(*-----*)  
(*-----*)
```

MODULE RESOLVE;

```
(*-----*)
(*)
(*) Author : Phil Dorr Created: 01-MAR-1988 (*)
(*) Version : 1.0 Revised: (*)
(*)
(*) Routine : Resolve (*)
(*)
(*) Function: (*)
(*) This routine will check for unresolved forward references (*)
(*) and print diagnostic information about them. (*)
(*)
(*) Environment: (*)
(*) VAX Pascal V3.3 and VMS 4.4 (*)
(*)
(*) Called by: (*)
(*) POST (via MAKXEQ) (*)
(*)
(*) Calls: (*)
(*) LTRIM, GETNAME, FINDKEY, MALGTK, MAEGKN, MALD (*)
(*) maeu,malno (*)
(*-----*)

(*--- Restrictions -----*)
(*)
None *)
(*-----*)

(*--- Commons -----*)
(*) See the POSTCOM include file documentation (*)
(*) See the V5INCLD include file documentation (*)

/7Reply received on MSEPD from user ZAID at _MSEPD$LTA734: 16:26:46
You have 15 min. to logoff before MSEPD goes down.Please

(*-----*)

(*--- Processing Description -----*)
(*) The user is notified that unresolved forward references exist (*)
(*) in the exchange format file. Then the unresolved reference (*)
(*) entities are processed. During this processing, the true entity (*)
(*) keys are located, and replaced into the appropriate constituent (*)
(*) lists. (*)
(*)
(*)
(*-----*)
```

```
(*--- Data Structures/Major Variables -----*)  
(*      Input  MASKEY      : entkey              *)  
(*      Input  ADB        : entblock             *)  
(*      Input  WORTHLESS   : blkdata             *)  
(*      Output IRC         : integer              *)  
(*-----*)
```

```
(*--- Change Control -----*)  
(*-----*)  
(*      Changed by:                               Date:      *)  
(*-----*)  
(*-----*)  
(*-----*)
```

MODULE RTRIM;

```
(*-----*)
(*)
(*) Author : Phil Dorr Created: 02-MAR-1988 (*)
(*) Version : 1.0 Revised: (*)
(*)
(*) Routine : RTRIM (*)
(*)
(*) Function: (*)
(*) This routine will take a varying character string and (*)
(*) remove any leading blanks. (*)
(*)
(*) Environment: (*)
(*) VAX Pascal V3.3 and VMS 4.4 (*)
(*)
(*) Called by: (*)
(*) WRTENT (*)
(*)
(*) Calls: (*)
(*) (none) (*)
(*)-----*)

(*--- Restrictions -----*)
(*)
(*) VAX Pascal can only handle strings with lengths less than (*)
(*) 65,535 characters. (*)
(*)
(*)-----*)

(*--- Commons -----*)
(*)
(*) No commons used. (*)
(*)
(*)-----*)

(*--- Processing Description -----*)
(*)
(*) Scan through the string looking for the last noblank char (*)
(*) Assign the substring of the input string, starting at the (*)
(*) first character and running to the last nonblank char to (*)
(*) the output string. (*)
(*)
(*)-----*)
(*--- Data Structures/Major Variables -----*)
(*)
```

CI PS560240032U
April 1990

```
(*   POS   The position in STR where the first noblank character   *)
(*       occurs.                                                    *)
(*   STR   The string to be trimmed.  This is passed to RTRIM by   *)
(*         reference.                                                *)
(*-----*)
```

```
(*--- Change Control -----*)
(*)
(*)   Changed by:                               Date:              *)
(*)
(*)-----*)
```

MODULE WRTENT;

```

(*)-----*)
(*)
(*) Author : P.D.Dorr Created: 3/3/88 (*)
(*) Version : 1.0 Revised: (*)
(*)
(*) Routine : WRTENT (*)
(*)
(*) Function: (*)
(*) The function of this routine is to write the mas entity (*)
(*) (passed in to this routine) out to the STEP exchange format file (*)
(*)
(*) Environment: (*)
(*) VAX Pascal V3.3 and VMS 4.4 (*)
(*)
(*) Called by: (*)
(*) CHECK (*)
(*)
(*) Calls: (*)
(*) CHECK (*)
(*) LTRIM (*)
(*) GETDD (*)
(*) GETID,PUTID (*)
(*) WRITEPACKED (*)
(*) malno,maecxq,maexeq,malgtk,maegkn,maeswt (*)
(*)
(*)-----*)

(*)--- Restrictions -----*)
(*) None (*)
(*)
(*)-----*)

(*)--- Files -----*)
(*) This routine accesses no external files (*)
(*)-----*)

(*)--- Commons -----*)
(*) See the PRECOM include file documentation (*)
(*)
(*)-----*)

(*)--- Processing Description -----*)
(*) Check to see if the entity's constituents have been (*)
(*) processed yet, if not then process them via an indirectly (*)
(*) recursive call to WRTENT. If the constituents have already been *)

```

```
(*      processed then simply write out each attribute of the entity to *)
(*      the STEP file according to its datatype.                          *)
(*-----*)
(*--- Data Structures/Major Variables -----*)
(* input variable MASKEY : entkey                                          *)
(* input variable ADB     : attribute data block                          *)
(* input variable DUMMY   : dummy argument required by MAEXEQ             *)
(* output variable IRC    : return code                                    *)
(*-----*)

(*--- Change Control -----*)
(* Changed by:                                     Date:                    *)
(*-----*)
```

SECTION 4

QUALITY ASSURANCE

4.1 Quality Assurance (QA) Requirements

The purpose of QA was to assure that GMAP software conforms to quality requirements and was developed in a cost effective manner consistent with good business practices. The QA function was established prior to contract award to interface with GMAP personnel during all phases of the project. Emphasis was placed on the detection and corrections of deviations from standards and on the detection and prevention of deficiencies in software.

Specific QA requirements are stated in the GMAP Software Quality Assurance (SQA) Plan which was developed by Pratt & Whitney and is included in the GMAP Technical Operating Report (TOR) (FR 19199-5). This plan was developed using Mil-S-52779A as a guide and was based on prior experience with Air Force projects. It states specific SQA requirements, including procedural requirements, in the following areas:

- o Software Development Management
- o Configuration Management
- o Reviews and Audits
- o Corrective Action
- o Library Control
- o Software Tools, Techniques, and Methodologies
- o Documentation
- o Testing.

In addition to these requirements, the SQA Plan established authority, organizational relationships, and responsibilities within the GMAP team. It required each subcontractor delivering GMAP software to submit its own SQA Plan adhering to the requirements of the overall GMAP plan.

These plans provided for specific guidelines and procedures to be used in all phases of software development, including the development of the System Test Plan (STP). The approach described above assured the maintainability, suitability, and general quality of software developed under the GMAP contract.